



freeBSD [®] JOURNAL

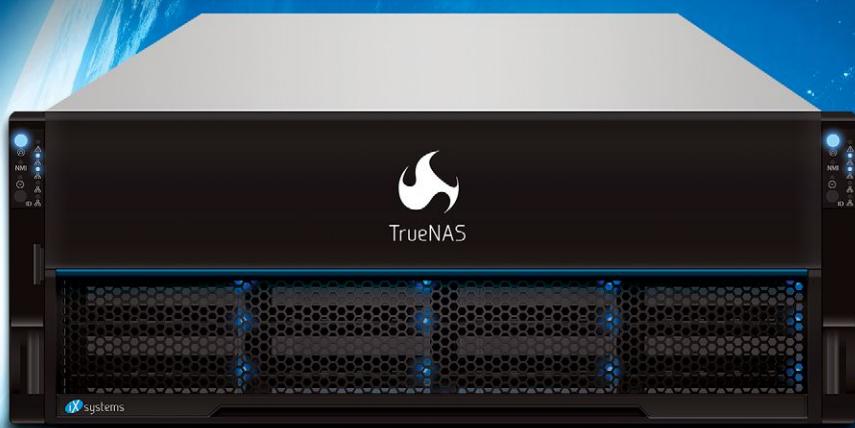
July/August 2018

Big Data

- **High-performance Computing**
and FreeBSD
- **Hadoop** *on FreeBSD*
with ZFS Tutorial
- *FreeBSD in*
Scientific Computing
- **pNFS**



DISRUPTING STORAGE AT WARP SPEED



LOWEST TCO SHARED STORAGE

Intel® Xeon® Scalable Family Processors

RESILIENT

- Self-healing Data
- Continuous Operation
- Easy Replication

WARP FACTOR IX

- Faster than SSD-based Hybrid Storage Arrays
- Flash-Turbocharged Data Access

EXPANDABLE

- Scales to 10PB using HGST Drives
- 10Gb/s-100Gb/s per NAS
- Non-disruptive Upgrades

COMPATIBLE

- Citrix, Veeam, & VMware Certified
- Unifies File, Block, & S3 Data
- Supports Leading Cloud Providers

Visit iXsystems.com/TrueNAS or call (855) GREP-4-iX today!





FreeBSD JOURNAL July/August 2018

TABLE OF CONTENTS

FreeBSD In

Scientific Computing

FreeBSD is an excellent platform for the needs of most typical scientific computing. If you mainly run open source software and prefer to spend your time doing science rather than IT maintenance, FreeBSD is definitely worth a try.

By Jason Bacon

4

10

Hadoop on FreeBSD with ZFS Tutorial

Running hadoop on FreeBSD has benefits. OpenZFS is able to add additional protection to the data stored in HDFS and makes it possible to store more data on the underlying disks. In a big data world, this is an enormous win.

By Benedict Reuschling

22

pNFS

A pNFS service separates the Read/Write operations from all other NFSv4.1 operations with the hope that this allows a pNFS service to be configured that exceeds the limits of a single NFS server for either storage capacity and/or I/O bandwidth. By Rick Macklem

30

High-performance Computing and FreeBSD

Where is the space for FreeBSD in the high-performance computing world and why should the FreeBSD community care about HPC? By Johannes M. Dieterich

36

NVM Express and FreeBSD

The authors describe the NVMe specification, FreeBSD's implementation of it and provide an overview of FreeBSD's utilities for monitoring and managing NVMe storage.

By Jim Harris and Warner Losh



3 Foundation Letter

By George Neville-Neil

42 Letters Kode Vicious suggested that Michael Lucas might be willing to handle a "Letters" column. This is the resulting correspondence! By Michael W Lucas

44 svn Update Digging through the subversion logs, I discovered that FreeBSD now supports pNFS, has a brand-new TCP congestion algorithm, and cron even gained new functionality!

By Steven Kreuzer

46 Events Calendar By Anne Dickison and Dru Lavigne

John Baldwin • FreeBSD Developer, Member of the FreeBSD Core Team and Co-Chair of *FreeBSD Journal* Editorial Board.

Brooks Davis • Senior Computer Scientist at SRI International, Visiting Industrial Fellow at University of Cambridge, and member of the FreeBSD Core Team.

Bryan Drewery • Senior Software Engineer at EMC Isilon, member of FreeBSD Portmgr Team, and FreeBSD Committer.

Justin Gibbs • Founder of the FreeBSD Foundation, Director of the FreeBSD Foundation Board, and a Software Engineer at Facebook.

Daichi Goto • Director at BSD Consulting Inc. (Tokyo).

Joseph Kong • Senior Software Engineer at Dell EMC and author of *FreeBSD Device Drivers*.

Steven Kreuzer • Member of the FreeBSD Ports Team.

Dru Lavigne • Director of Storage Engineering at iXsystems, author of *BSD Hacks* and *The Best of FreeBSD Basics*.

Michael W Lucas • Author of *Absolute FreeBSD*.

Ed Maste • Director of Project Development, FreeBSD Foundation.

Kirk McKusick • Treasurer of the FreeBSD Foundation Board, and lead author of *The Design and Implementation* book series.

George V. Neville-Neil • President of the FreeBSD Foundation Board, and co-author of *The Design and Implementation of the FreeBSD Operating System*.

Philip Paeps • Secretary of the FreeBSD Foundation Board, FreeBSD Committer, and Independent Consultant.

Hiroki Sato • Director of the FreeBSD Foundation Board, Chair of Asia BSDCon, member of the FreeBSD Core Team, and Assistant Professor at Tokyo Institute of Technology.

Benedict Reuschling • Vice President of the FreeBSD Foundation Board, a FreeBSD Documentation Committer, and member of the FreeBSD Core Team.

Robert N. M. Watson • Director of the FreeBSD Foundation Board, Founder of the TrustedBSD Project, and University Senior Lecturer at the University of Cambridge.

S&W PUBLISHING LLC

PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Copy Editor • Annaliese Jakimides

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
5757 Central Ave., Suite 201, Boulder, CO 80301
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

Copyright © 2018 by FreeBSD Foundation. All rights reserved. This magazine may not be reproduced in whole or in part without written permission from the publisher.

Perhaps not all our readers may know this but the Berkeley Software Distribution, BSD, from which FreeBSD is descended, started out as, essentially, a science project. Since the early days of BSD at UC Berkeley our systems have often been used in the advancement of science, whether on computer science or in support of non-computer science, such as physics, chemistry and biology. Many of the technologies that are now mainstream, such as virtual memory and the TCP/IP protocols, started out as experiments in BSD. Long before the term machine learning was coined, the BSDs were used to store and process scientific data at universities around the world, and in the National Laboratories of the US and other countries. In our latest issue of the *FreeBSD Journal*, we talk about modern FreeBSD in scientific computing, with articles from Jason Bacon, Benedict Reuschling, and Johannes Dieterich.

We round out our issue with two articles on storage, one on NVM Express, from Jim Harris and Warner Losh, who have written a good chunk of the code in FreeBSD that handles this new, persistent, high speed storage technology. Rick Macklem takes us through pNFS, which allows the Network File System (NFS) to operate in parallel, across a series of servers, which is a common use case within the scientific computing community.

Finally, we have our usual collection of columns this month, with Steven Kreuzer discussing the commit of pNFS to FreeBSD in *svn Update*, Michael Lucas handling our Letters, and Dru Lavigne keeping you up-to-date on all that is coming up in future months with the Events Calendar.

All of us here at the *Journal* hope that if you're in a place that's quite summery, you'll pour a version of your most relaxing beverage, sit in the shade, and enjoy this cool mix of all that's best in FreeBSD.

George Neville-Neil
President of the FreeBSD Foundation Board of Directors



FreeBSD IN SCIENTIFIC COMPUTING

By Jason Bacon

I've been running both FreeBSD and Linux uninterrupted since the mid-1990s. I've run many different Linux distributions over the years, most recently focusing on CentOS. I also have significant experience with Mac OS X and NetBSD and have experimented with many other BSD platforms. As a staunch agnostic with a firm belief in the value of open standards, I like to remain familiar with all the options in the POSIX world so I'm always prepared to choose the best tool for the job.

Over the years, I've watched barriers to FreeBSD use fall one by one. Open-source software continues to spread like the Blob, filling in niches once held exclusively by commercial and other closed-source software.

OpenOffice/LibreOffice, OpenJDK, Clang, Flang, and many other high-quality open-source products have made it possible for most people to do everything they need on FreeBSD with ease. The few remaining limitations of what can be run on FreeBSD have become increasingly esoteric and they, too, seem destined to disappear in time.

My need to run MS Windows for personal use came to an end more than a decade ago, although I still support it to some extent for work. Mac OS X has filled the few remaining needs that had been served by Windows, but even my Mac has now been reduced to a once-per-year platform for running tax software. I could, of course, use FreeBSD for this as well with a web-based tax program, if not for the reality of computer security in the cloud. The bottom line is that I now find FreeBSD easier and more pleasant to use than commercial operating systems for most of my work and personal computing.

Most of my professional work since late 1999 has been in support of scientific computing. From 1999 until 2008, I supported fMRI brain-mapping research for a multidisciplinary group including neu-

rologists, neuropsychologists, cell biologists, psychiatrists, and biophysicists. During most of this time, I was the sole IT support person for several labs, peaking at over 60 researchers. I was responsible for maintaining many Unix workstations as well as managing all the research software needed for fMRI analysis. The need for a full-featured and extremely reliable operating system became very clear, very quickly. FreeBSD answered that call and made it possible for me to single-handedly keep this important research moving forward for many years.

Since 2009, I have been supporting a wide range of research, including engineering, bioinformatics, physics, math, chemistry, public health, business, and psychology. A major part of this environment has been our HPC clusters running CentOS Linux. FreeBSD has also played an important role as a development and testing platform, and the primary OS on our educational HPC cluster and HTCondor grid. It has also provided the model for how we manage most open-source software on our CentOS systems, using `pkgsrc`, a cross-platform package manager from the NetBSD project, originally derived from FreeBSD ports.

Already an Important Part of HPC

FreeBSD already plays crucial roles in research, including many high-performance computing (HPC) environments, although some may not rec-

ognize it by name. FreeBSD is the foundation of some of the most popular high-performance storage appliances such as FreeNAS, Isilon, NetApp, and Panasas. FreeBSD is also an important component of Juniper network switches, pfSense firewalls, Apple's OS X, and runs servers for many web-hosting and cloud services. FreeBSD enjoys strong support from other vendors as well, including Mellanox, which is currently developing FreeBSD drivers for their network adapters.

Outside scientific research, FreeBSD is used by some of the biggest players in the business, who need to maximize performance and reliability. Ever watch a movie on Netflix? If so, you're a FreeBSD user. The Netflix servers streaming movies to your TV or computer run FreeBSD. FreeBSD has also been used for Yahoo! servers for decades. For a current list of notable companies using FreeBSD, see the FreeBSD Handbook (https://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/nutshell.html#introduction-nutshell-users).

FreeBSD as a Computing Platform

Scientific computing has reached a sort of utopia in recent years due to fast, cheap hardware and full-featured, free software. For a typical workstation or laptop, it's hard for a scientist to go wrong. There are multiple free operating systems to choose from, mostly based on BSD, Linux, and Solaris. Any one of them will serve the needs of a typical scientist quite well. There are POSIX compatibility layers for MS Windows such as Cygwin and Windows Subsystem for Linux (WSL), which facilitate running Unix software directly on Windows, and free virtual machines, such as VirtualBox and qemu, that allow us to run multiple operating systems at the same time on the same machine.

Unsurpassed Reliability

But what about those of us who manage many multi-user servers for research? There has always been, and always will be, a severe shortage of skilled systems managers. Nowhere is this shortage felt more painfully than in scientific research. The law of supply and demand dictates that experienced systems managers are expensive. Scientific researchers make their living by repeatedly begging for grant money to keep their labs running. With few exceptions, most researchers have too little funding and cannot hope to compete with wealthy corporations for the limited IT talent pool. In fact, most principle investigators in academic research earn far less than an experienced Unix sysadmin.

In this scenario, choosing a system that minimizes IT man-hours is crucial. We must enable a small IT

staff to be as productive as possible, by avoiding system outages and performing common tasks such as software deployment as quickly and cleanly as possible. This is where FreeBSD stands out among its peers. FreeBSD's unsurpassed reliability and ease of management save precious IT man-hours that could be used for more creative endeavors. It can literally reduce the need for man-hours by an order of magnitude over the methods often used by inexperienced sysadmins in the research community.

Experienced systems managers in this situation know to stay away from bleeding-edge operating systems, which are likely to bring repeated surprises that will distract them from more creative work. For this reason, the vast majority of large HPC clusters run Enterprise Linux rather than other distributions with the latest kernel, compilers, and other system software. This is not a criticism of bleeding-edge platforms. In fact, it's important to all of us that many typical users use them and work the bugs out of the latest new kernel features, compilers, etc. Today's bleeding-edge is tomorrow's Enterprise.

Enterprise Linux systems achieve a high level of reliability and long-term binary compatibility by taking a snapshot of a recent bleeding-edge system and declaring a moratorium on major upgrades. For example, Redhat Enterprise and CentOS are based on a snapshot of Fedora. The down side of this approach is that the tools and libraries that ship with it are outdated and unable to support the latest open-source scientific software. For example, Redhat Enterprise 7, the latest release as of this writing, ships with GCC 4.8.5 (June 2015) and GNU libc 2.17. The latest releases are GCC 8.1 and libc 2.27. The only way to build the latest open source on such a system is by installing newer core tools and libraries. Replacing them outright would sacrifice the stability and binary compatibility with commercial software for which Enterprise Linux is designed. The solution we employ on our CentOS systems is to leave all these core tools in-place and use pkgsr, a cross-platform package manager, to install newer tools alongside them. Many others have resorted to using containers or virtual machines to provide a more modern environment, isolated from the outdated Enterprise base.

FreeBSD allows us to avoid these issues by offering stability matching or exceeding that of Enterprise Linux while providing more modern tools in the base. As of this writing, FreeBSD 11.2 and the upcoming FreeBSD 12 release both provide clang 6 (March, 2018) as a base compiler and easily support most current open-source scientific software.

Minimize Management Time

A FreeBSD system can be installed from scratch in about 5 minutes and fully configured for many

purposes in less than an hour, thanks to well-designed tools in the base system and the FreeBSD ports system for managing add-on packages (more on this later). The FreeBSD handbook is a well-written and excellent tutorial for most common tasks and is kept fairly up-to-date. No need to search the web and risk following outdated or erroneous instructions.

Ongoing maintenance is quick and easy using freebsd-update, a binary update system for installing bug fixes and security patches in the base system. Unlike many other operating systems, FreeBSD security notices include clear instructions on how to apply patches, including when a reboot or service restart is required.

Software Management with FreeBSD Ports

To go from idea to a published paper in scientific computing involves four steps:

1. Develop the software
2. Deploy the software
3. Learn the software
4. Run the software

Steps 1 and 3 are mostly platform-independent. Unfortunately, step 2, software deployment, is often one of the major bottlenecks in scientific computing. Step 4, running the software, is a lesser, but real bottleneck requiring the ability to deploy an optimized build of the software. FreeBSD ports has the potential to easily eliminate these bottlenecks, as it allows the user to easily install from a huge collection of binary packages and just as easily install any package from source with additional compiler options. FreeBSD also has strong support for OpenMP, pthreads, and MPI, for cases where parallel computing is needed to reduce run times.

The vast majority of scientific software is open source, developed on a variety of (usually bleeding-edge) platforms, and often deployed via very primitive methods. Many developers don't target package managers at all, but instead provide precompiled binaries for their own development platform and maybe a few others, alongside cryptic instructions for performing a "caveman" install, manually building from source after installing dependencies, or worse, using their bundled dependency software. The chances of their build-from-source instructions working for the average user are almost nil.

Part of the problem is that the developers are mostly scientists with little or no computer science training, very often self-taught graduate students developing software for their dissertation. They don't know much about sustainable development and systems management practices. If their software lives beyond their graduation date, it will likely get

cleaned up so that it's more portable and easier to deploy. However, given the constant, rapid progress of science and the rotating door of student-developers, the research computing community is basically doomed forever to a world full of nascent, disorganized code.

The FreeBSD ports system can alleviate this problem in two ways:

FreeBSD ports has one of the largest collections of existing packages of any package manager. At the time of this writing, FreeBSD users can install any of more than 32,000 packages with one simple command. If the package you need is not already in the collection, chances are that most or all the dependencies are there, so the effort needed to create a new FreeBSD port is often a small fraction of that required to do a caveman install. Note also that FreeBSD's port options substantially increase the number of possible software installations. This has to be considered when comparing the ~32,000 FreeBSD ports against the number of binary packages in systems that do not support convenient builds from source. Many of the binary packages in such systems are merely different builds of the same software.

Imagine a scenario where instead of thousands of scientists each wasting forty hours struggling with the same caveman installation, one of them creates a FreeBSD port and everyone else in the world from that day on can install the software in seconds. That's many thousands of man-hours redirected from senseless, duplicated IT effort to productive scientific exploration. This is the potential of FreeBSD ports and other package managers.

There is another important difference between binary package managers, which quickly install pre-compiled packages along with dependencies, and package managers that make it convenient to build from source, such as FreeBSD ports, Gentoo Portage, MacPorts, and pkgsrc. Binary packages install much faster, of course, but they may suffer from compatibility, security, and performance problems. To be portable, they must be compiled with static libraries and limited to common CPU features, which precludes utilizing new instructions and other CPU features that may significantly improve performance. In extreme cases, you may see a 30% improvement in speed from a non-portable binary utilizing all available CPU features. This can save thousands of core-hours on an HPC cluster running a large analysis.

With FreeBSD ports, building an optimized binary from the source code, utilizing the best features of your CPU, is as easy as installing the binary package. It will take longer for the computer to build and install, of course, but the effort for you is about the same. To install a portable (possibly slow) binary package, we might use the following:

```
pkg install canu
```

To build an optimized version from source, we would adjust our build settings in `/etc/make.conf` (e.g. by adding `CXXFLAGS+=-march=native`), and run the following:

```
cd /usr/ports/biology/canu
make install
```

If a FreeBSD port does not already exist for the software you need, consider creating one. It's not as difficult as one might assume. There is a rather steep learning curve to becoming a FreeBSD ports committer, who can add ports to the system only after extensive quality control measures.

However, ports need not be committed before you can use the ports system to deploy them. In fact, all ports are deployed and tested before being committed. The learning curve for creating a basically functional port or upgrading an existing port is fairly small. Anyone who knows how to write a Makefile and use the build system employed by the upstream developers can learn to do this fairly quickly. The Porter's Handbook (https://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook/) covers most of what you would need to know, starting from the beginner level all the way to becoming a FreeBSD committer.

If you install binary packages from the FreeBSD ports system, they can quickly and easily be updated using the following command:

`pkg upgrade`

(Note that this may replace your optimized from-source install with a newer binary package, so watch for this and rebuild the port from source after the `pkg upgrade` if necessary.)

The port frameworks used to build from source can also be easily updated using `portsnap` or `svn`. This is a great feature for those who want to keep their systems running the latest of everything. But what if you need to keep the same version of a program running through a long-term study spanning several months or even years? Running `pkg upgrade` could effectively break your study.

It is possible, though not well-tested at this stage, to deploy multiple ports trees under different prefixes. Ports can also be installed this way without root privileges. The FreeBSD ports project branches snapshots every three months under different prefixes. The ports in these quarterly snapshots are never upgraded, although they may receive bug and security patches.

I have been experimenting with deploying quarterly snapshots under prefixes such as `/sharedapps/ports-2018Q1`, with corresponding installation to `/sharedapps/local-2018Q1`. Software can be installed statically here and never upgraded, while other software installed to the standard prefix is upgraded regularly with `pkg upgrade`.



RootBSD
Part of the NetActuate family.

Leverage Our Global Footprint of 32 Locations Worldwide

- ✓ Deploy FreeBSD instantly via our web portal
- ✓ Cloud servers and colocation in all major global markets
- ✓ Customized, dedicated bare metal available
- ✓ Full root access
- ✓ 24x7 friendly support from FreeBSD experts on staff

Request a quote and learn more today at
netactuate.com

This system follows a better-supported feature of the `pkgsrc` package manager, which is designed to be bootstrapped on any POSIX platform under any prefix the user desires. We have been using `pkgsrc` this way on our CentOS systems for years.

We could also use `pkgsrc` this way on FreeBSD, but there is a strong motivation to use the FreeBSD ports in a similar fashion, mainly because it has a larger collection than `pkgsrc` at this time. Some work remains to be done to utilize FreeBSD ports this way, but the basic support for this sort of deployment already exists. We just need to put it into use and iron out the wrinkles.

Base Features Beneficial to Science

There is a strong interest in the advanced ZFS filesystem in the scientific community. Its performance, flexibility, and data protection features are very attractive to users who invest immense amounts of time and money generating files containing their research results.

FreeBSD's RootOnZFS features allow us to deploy a FreeBSD installation booting from a ZFS filesystem using a simple menu interface in the installer. Any modern PC with multiple disks can be up and running with FreeBSD on a RAIDZ array in a matter of minutes.

ZFS is not suited for every purpose, however. ZFS is rather memory-hungry. On an HPC compute node, where local disks are used only to house the operating system and provide temporary storage, we may not want ZFS competing for memory resources with the computational processes. The same reasoning would apply to any other machine devoted mainly to CPU- or memory-bound tasks. Fortunately, FreeBSD's UFS2 filesystem also provides solid performance and reliability, with a very low memory footprint.

FreeBSD provides enterprise reliability and easy management, but what about that other big advantage of Enterprise Linux, support for commercial software products? Fortunately, FreeBSD's Linux compatibility module allows us to run most Linux binaries, *with no performance penalty*, trivial memory overhead, and a very modest amount of disk space and effort. FreeBSD's Linux compatibility is often erroneously referred to as *emulation* or a *compatibility layer*.

In reality, it is neither. The basis of FreeBSD's Linux compatibility is a kernel module that *directly* supports Linux system calls. The module activates an alternative function pointer table to directly invoke Linux-compatible kernel functions when a Linux binary is being run.

To complete the Linux-compatible environment, we simply need to install the Linux versions of any shared libraries and tools required by Linux programs, the same as we would on a real Linux system. The

FreeBSD ports system provides tools for easily installing RPMs used by the RHEL/CentOS Yum package manager. Creating a FreeBSD port that installs software from the CentOS Yum repositories is trivial in most cases, and many such ports you might need already exist.

FreeBSD's Linux compatibility is fairly robust and capable of running the vast majority of commercial Linux binaries. I have personally run many versions of Linux Matlab on FreeBSD machines, with full functionality, including the Java desktop and MEX compilation system (using Linux GCC compilers).

At this point, though, I would advise most FreeBSD users to run Octave, a full-featured, open-source Matlab-compatible suite. It's virtually identical to Matlab for most typical users, it's free, and can be installed in seconds.

There is, of course, some additional work required to run Linux binaries on FreeBSD. Running something as complex as Matlab or ANSYS may require a fair amount of effort. Simpler applications such as shared-memory LS-DYNA are trivial to install. Linux binaries depending on MPI (Message Passing Interface) for distributed parallel computing could also be a challenge. If you rely heavily on complex closed-source Linux applications, you may be better off running an Enterprise Linux system.

In order for FreeBSD to become a competitive platform in HPC, it would also need to more easily support a few other subsystems that currently only work well on Linux, such as nVidia's CUDA GPU platform (although the open standard OpenCL is beginning to gain traction) and Infiniband interconnects that are commonly used for distributed parallel computing.

There are a few remaining features that are likely to ensure the dominance of Enterprise Linux in HPC for a while.

As it stands, though, FreeBSD is already an excellent platform for the needs of most typical scientific computing. If you mainly run open source software and prefer to spend your time doing science rather than IT maintenance, FreeBSD will serve you well. ●

The lead sysadmin for research computing at the University of Wisconsin-Milwaukee, Jason Bacon has been working with computers since 1983, and with FreeBSD and Linux since 1995. He is the author of *The C/Unix Programmer's Guide*, a comprehensive guide for beginning to intermediate C/C++ programming under UNIX, Linux, Macintosh OS X, and similar systems. He is proficient in Spanish and German, with a working knowledge of French and Mandarin. When he isn't inside working on various systems, he can often be found outside, cycling, sea kayaking, cross-country skiing, hiking, and scuba diving.

Support FreeBSD®



Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsdfoundation.org/donate

Hadoop

on FreeBSD with ZFS

Tutorial

By Benedict **Reuschling**

This article provides a scripted tutorial using an Ansible playbook to build the cluster from the ground up. Manual installation and configuration quickly becomes tedious the more nodes are involved, which is why automating the installation removes some of the burden for BSD system administrators. The tutorial builds hadoop on ZFS to make use of the powerful features of that filesystem with an integrated volume manager to enhance HDFS's feature set.

.....

Hadoop is an open-source, distributed framework using the map-reduce programming paradigm to split big computing jobs into smaller pieces. Those pieces are then distributed to all the nodes in the cluster (map step), where the participating datanodes calculate parts of the problem in parallel. In the reduce step, those partial results are collected and the final result is computed. Results are stored in the hadoop distributed filesystem (HDFS). Coordination is done via a master node called the namenode. Hadoop consists of many components and can run on commodity hardware without requiring many resources. The more nodes that participate in the cluster and if the problem can be expressed in map-reduce terms, the better the performance than just running the calculation on a single node. Mostly written in Java, hadoop aims to provide enough redundancy to allow nodes to fail while still maintaining a functional compute cluster. A rich ecosystem of additional software

grew up around hadoop, which makes the task of setting up a cluster of hadoop machines for big data applications a difficult one.

Requirements

This tutorial uses three FreeBSD 11.2 machines, either physical or virtual. Other and older BSD versions should work as well, as long as they support a recent version of Java/OpenJDK. OpenZFS with regular directories is fine, If OpenZFS is not available, regular directories are fine, too. One machine will serve as the master node (called namenode) and the other two will serve as compute nodes (or datanodes in hadoop terms). They need to be able to connect to each other on the network. Also, an Ansible setup must be available for the playbook work. This involves an inventory file that contains the three machines and the necessary software on the target machines (python 2.7 or higher) for Ansible to send commands to them.

Note that this playbook does not use the default paths used by the FreeBSD port/package of hadoop. This way, a higher version of hadoop can be used before the port gets updated. The default FreeBSD paths can be easily substituted when required. The configuration files presented in this tutorial contain only the minimal sections required to get a basic hadoop setup going. The FreeBSD port/package contains sample configuration files that have many more configuration options than are initially needed. However, the port is a great resource for extending and learning about the hadoop cluster once it is set up.

Readers unfamiliar with Ansible should be able to abstract the setup steps and either implement them with a different configuration management system (puppet, chef, saltstack) or execute the steps manually.



Defining Playbook Variables

To make management easier, a separate `vars.yml` file that holds all the variables is used. This file contains all the information in a central location that is needed to install the cluster. For example, when a higher version of hadoop should be used, only the `hdp_ver` variable must be changed.

```
java_home: "/usr/local/openjdk8"
hdp: "hadoop"
hdp_zpool_name: "{{hdp}}pool
hdp_ver: "2.9.0"
hdp_destdir: "/usr/local/{{hdp}}/{{hdp_ver}}"
hdp_home: "/home/{{hdp}}"
hdp_tmp_dir: "/{{hdp}}/tmp"
hdp_name_dir: "/{{hdp}}/hdfs/namenode"
hdp_data_dir: "/{{hdp}}/hdfs/datanode"
hdp_mapred_dir: "/{{hdp}}/mapred.local.dir"
hdp_namesec_dir: "/{{hdp}}/namesecondary"
hdp_keyname: "my_hadoop_key"
hdp_keytype: "ed25519"
hdp_user_password: "{{vault_hdp_user_pw}}"
```

File: `vars.yml`

The first line stores the location of the installed OpenJDK from ports. To save a bit of typing in the playbook and to replace common occurrences of the word `hadoop`, a shorter variable `hdp` is used as a prefix to all the rest of the variables. The zpool name (`hadooppool` in this tutorial) already makes use of the variable we defined for `hadoop`'s name. As mentioned above, the `hadoop` version is used to keep track of which version this cluster is based on. The variables describe ZFS datasets (or directories) for the `hadoop` user that the software is using while running. The last couple of lines are defining the SSH key that `hadoop` needs to securely connect between the nodes of the cluster. Secrets like passwords are also stored for the `hadoop` user in `Ansible-vault`. This way, the playbook can be shared with others without exposing the passwords set for that individual cluster.

To create a new vault, the `ansible-vault(1)` command with the `create` subcommand is used, followed by the path where the encrypted vault file should be stored.

```
$ ansible-vault create vault.yml
```

After being prompted to create a passphrase to open the vault, an editor is opened in the vault file and secrets can be stored within. Refer to (https://docs.ansible.com/ansible/latest/reference_appendices/faq.html#how-do-i-generate-crypted-passwords-for-the-user-module) on how to generate encrypted passwords that the `Ansible` user module can understand. The line in the vault should look like this, with `<password>` replaced by the password hash:

```
vault_hdp_user_pw: "<yourpassword>"
```

File: `vault.yml`

Playbook Contents

The playbook itself is divided into several sections to help better understand what is being done in each of them. The first part is the beginning of the playbook where it describes what the playbook does (`name`), which hosts to work on (`hosts`), and where the variables and the vault are stored (`vars_files`):

```
#!/usr/local/bin/ansible-playbook
- name: "Install a {{hdp}} {{hdp_ver}} multi node cluster"
  hosts: "{{host}}"

  vars_files:
  - vault.yml
  - vars.yml
```

The first line will ensure that the playbook can run like a regular shell script by making it executable (`chmod +x`). The `name:` describes what this playbook is doing and uses the variables defined in `vars.yml`. The hosts are provided on the commandline later to make it more flexible to add more machines. Alternatively, when there is a predetermined number of hosts for the cluster, they can also be entered in the `hosts:` line.

Next, the tasks that the playbook should execute are defined (be careful not to use tabs for indentations, this is YAML syntax):

```
tasks:
- name: "Install required software for {{hdp}}"
  package:
    name: "{{item}}"
  with_items:
    - openjdk8
    - bash
    - gtar
```

The first task is to install OpenJDK from FreeBSD packages, bash for the hadoop user's shell, and gtar to extract the source tarball (the `unarchive` step later on) that was downloaded from the hadoop website.

The datasets (or directories if ZFS can not be used) are created in the next step:

```
- name: "Create ZFS datasets for the {{hdp}} user"
  zfs:
    name: "{{hdp_zpool_name}}{{item}}"
    state: present
    extra_zfs_properties:
      mountpoint: "{{item}}"
      recordsize: "1M"
      compression: "lz4"
  with_items:
    - "{{hdp_home}}"
    - "/opt"
    - "{{hdp_tmp_dir}}"
    - "{{hdp_name_dir}}"
    - "{{hdp_data_dir}}"
    - "{{hdp_namesec_dir}}"
    - "{{hdp_mapred_dir}}"
    - "{{hdp_zoo_dir}}"
```

The datasets are each using LZ4 for compression and are able to use a record size of up to 1 megabyte. This is important to increase compression as the hadoop distributed filesystem (HDFS) is using 128MB records by default. The paths to the mount points are defined in the `vars.yml` file and will be used in the hadoop-specific config files again later on.

```

- name: "Create the {{hdp}} User"
  user:
    name: "{{hdp}}"
    comment: "{{hdp}} User"
    home: "{{hdp_localhome}}/{{hdp}}"
    shell: /usr/local/bin/bash
    createhome: yes
    password: "{{vault_hdp_user_pw}}"

```

The hadoop processes should all be started and run under a separate user account, aptly named hadoop. This task will create that designated user in the system. The result looks like the following in the password database (the user ID might be a different one):

```

$ grep hadoop /etc/passwd
hadoop:*:31707:31707:hadoop User:/home/hadoop:/usr/local/bin/bash

```

Next, the SSH keys need to be distributed for the hadoop user to be able to log into each cluster machine without requiring a password. Ansible's lookup functionality is used to read an SSH key that was generated earlier on the machine running the playbook (it is recommended to generate this kind of separate key for hadoop using `ssh-keygen`). The SSH key must not have a passphrase, as the hadoop processes will perform the logins without any user interaction to enter it. The task will add the SSH public key to the `authorized_keys` file in `/home/hadoop`.

```

- name: "Add SSH key for {{hdp}} User to authorized_keys file"
  authorized_key:
    user: "{{hdp}}"
    key: "{{ lookup('file', './{{hdp_keyname}}.pub') }}"

```

The public and private key must be placed in hadoop's home directory under `.ssh`. Since a variable has been defined for the key, it is easy to provide the public (`.pub` extension) as well as the private key (no extension) without having to spell out its real name in this task. Additionally, the key is secured by setting a proper mode and ownership so that no one else but hadoop has access to it.

```

- name: "Copy public and private key to {{hdp}}'s .ssh directory"
  copy:
    src: "./{{item.name}}"
    dest: "{{hdp_localhome}}/{{hdp}}/.ssh/{{item.type}}"
    owner: "{{hdp}}"
    group: "{{hdp}}"
    mode: 0600
  with_items:
    - { type: "id_{{hdp_keytype}}", name: "{{hdp_keyname}}" }
    - { type: "id_{{hdp_keytype}}.pub", name: "{{hdp_keyname}}.pub" }

```

The hadoop user is added to the `AllowUsers` line in `/etc/ssh/sshd_config` to allow it access to each machine. The regular expression will make sure that any previous entries in the `AllowUsers` line are preserved and that the hadoop user is added to the end of the preexisting user list.

```

- name: "Add {{hdp}} to AllowedUsers line in /etc/ssh/sshd_config"
  replace:
    backup: no
    dest: /etc/ssh/sshd_config
    regexp: '^(AllowUsers(?!.*\b{{ hdp }}\b).*)$'
    replace: '\1 {{ hdp }}'
    validate: 'sshd -T -f %s'

```

SSH is restarted explicitly afterwards, as the playbook is going to make use of the hadoop SSH login soon. Note that an Ansible handler can't be used here, because it would be executed too late (at the end of the playbook when all tasks have been executed).

```
- name: Restart SSH to make changes to AllowUsers take effect
  service:
    name: sshd
    state: restarted
```

The next task deals with collecting SSH key information from the node so that hadoop does not have to confirm the host key of the target system upon establishing the first connection. We need to be able to locally ssh into the master node itself, so we have to add `0.0.0.0, localhost`, the IP address of each machine, the master IP address (so that the client nodes know about it and don't require an additional task) to `.ssh/known_hosts`. That is what `ssh-keyscan` is doing in this task step. The variable `{{workers}}` will be provided on the commandline later and contains all the machines that will act as datanodes to run map-reduce jobs. (Of course, these can also be placed in `vars.yml` when the number of machines is static and do not change.)

```
- name: "Scan SSH Keys"
  shell: ssh-keyscan 0.0.0.0 localhost \
"{{hostvars[inventory_hostname]:'ansible_default_ipv4'}['address']}} {{master}}" >>
"{{hdp_home}}/.ssh/known_hosts"

- name: "Scan worker SSH Keys one by one"
  shell: "ssh-keyscan {{item}} {{master}} >> {{hdp_home}}/.ssh/known_hosts"
  with_items: "{{workers}}"
```

To function properly, hadoop requires setting a number of environment variables. These include `JAVA_HOME`, `HADOOP_HOME` and other variables that the hadoop user needs to make the hadoop cluster work with Java. The environment variables are stored in the `.bashrc` file that is deployed from the local Ansible control machine to the hadoop home directory on the remote systems.

The `.bashrc` file itself will be provided as a template. This powerful functionality in Ansible makes it possible to store configuration files filled with Ansible variables (utilizing Jinja2 syntax). When deploying them, it is not just a simple copy operation. During transport to the remote machine, the variables are replaced with their actual values. In this case, `{{hdp_destdir}}` is replaced by `/usr/local/hadoop2.9.0`.

```
- name: "Copy BashRC over to {{hdp_localhome}}/{{hdp}}/.bashrc"
  template:
    src: "./hadoop.bashrc.template"
    dest: "{{hdp_home}}/.bashrc"
    owner: "{{hdp}}"
    group: "{{hdp}}"
```

The template file itself needs to have the following additional content at the end of the file:

```
export JAVA_HOME={{java_home}}
export HADOOP_HOME={{hdp_destdir}}
export HADOOP_INSTALL={{hdp_destdir}}
export PATH=$PATH:$HADOOP_INSTALL/bin
export HADOOP_PREFIX=/opt/{{hdp}}
export PATH=$PATH:$HADOOP_PREFIX/bin
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
export HADOOP_MAPRED_HOME=$HADOOP_HOME
```

```

export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="$HADOOP_OPTS -Djava.library.path=$HADOOP_HOME/lib/native"

```

Any users other than hadoop that want to run map-reduce jobs would also need these environment variables set, so consider also copying that file to `/etc/skel/.bashrc`.

Deploying Hadoop Configuration Files

It is time to deploy the files that make up the hadoop distribution. It is basically a tarball that can be extracted to any directory, as it mostly contains JARs and config files. This way, hadoop can easily be copied around as a whole, since the directory contains everything needed to run hadoop. The files are available for download from the hadoop webpage (<http://hadoop.apache.org/releases.html>). There are a lot of supported versions that keep evolving rapidly, meaning that there will be new releases coming out at regular intervals. The bottom of that page lists how many bugs were fixed in each of the releases. Contrary to how it might sound, there is no need to keep up with the pace that the hadoop project sets and an older release of hadoop can run for years if desired. A fairly recent release (2.9.0) was chosen for this article. Make sure to pick the binary distribution to download, as it takes additional time to build hadoop from sources. The file is called `hadoop-2.9.0.tar.gz`, and the name will be constructed again in the playbook by using the definitions in the `vars.yml` file. Ansible's `unarchive` module takes care of extracting the tarball on the remote machine into `{{hdp_destdir}}`, which resolves to `/usr/local/hadoop2.9.0`. With the version included in the directory/dataset name, it is possible to install different versions of hadoop side by side for testing purposes.

```

- name: "Unpack Hadoop {{hdp_ver}}"
  unarchive:
    src: "./{{hdp}}-{{hdp_ver}}.tar.gz"
    dest: "{{hdp_destdir}}"
    remote_src: yes
    owner: "{{hdp}}"
    group: "{{hdp}}"

```

Core Hadoop Configuration

The time has come to edit the fleet of configuration files that ship with hadoop. It can be overwhelming for beginners starting out with hadoop to understand which file needs to be changed. Unfortunately, the hadoop website does not do a good job of explaining what files need to be changed for a fully distributed hadoop cluster. In our experience, the documentation on the hadoop website is incomplete, and, even if followed to the letter, the result is not a functioning hadoop cluster. After a lot of trial and error, the author identified the important files needed to create a fully functional map-reduce cluster with the underlying HDFS on FreeBSD.

At its core, 4 files form the site-specific configuration parts for this cluster and they are named `*-site.xml`. They need to be changed and all of them reside in the configuration directory of the hadoop distribution. In this tutorial, that path is `/usr/local/hadoop2.9.0/etc/hadoop/` and contains `core-site.xml`, `yarn-site.xml`, `hdfs-site.xml`, and `mapred-site.xml`.

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://{{master}}:9000</value>
  </property>
  <property>

```

```
<name>io.file.buffer.size</name>
<value>131072</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>{{hdp_tmp_dir}}</value>
  <description>A base for other temporary directories.</description>
</property>
</configuration>
```

core-site.xml Template

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <!-- Site specific YARN configuration properties -->
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>{{master}}</value>
  </property>
  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>{{master}}:8025</value>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>{{master}}:8030</value>
  </property>
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>{{master}}:8050</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>0.0.0.0:8088</value>
  </property>
</configuration>
```

yarn-site.xml Template

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file://{{hdp_name_dir}}</value>
  </property>
```

```

<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///{{hdp_data_dir}}</value>
</property>
<property>
  <name>dfs.replication</name>
  <value>2</value>
</property>
<property>
  <name>dfs.checkpoint.dir</name>
  <value>file:///{{hdp_freebsd_namesec_dir}}</value>
  <final>true</final>
</property>
</configuration>

```

hdfs-site.xml Template

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>mapreduce.job.tracker</name>
    <value>{{inventory_hostname}}:8021</value>
  </property>
  <property>
    <name>mapred.job.tracker</name>
    <value>{{inventory_hostname}}:54311</value>
  </property>
  <property>
    <name>mapred.local.dir</name>
    <value>{{hdp_mapred_dir}}</value>
  </property>
  <property>
    <name>mapred.system.dir</name>
    <value>/mapredsystemdir</value>
  </property>
  <property>
    <name>mapred.tasktracker.map.tasks.maximum</name>
    <value>2</value>
  </property>
  <property>
    <name>mapred.tasktracker.reduce.tasks.maximum</name>
    <value>2</value>
  </property>
  <property>
    <name>mapred.child.java.opts</name>
    <value>-Xmx200m</value>
  </property>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.address</name>

```

```

<value>{{inventory_hostname}}:10020</value>
</property>
</configuration>

```

mapred-site.xml Template

The following task in the playbook will take care of putting them in the right place with properly replaced variables from the `vars.yml` definition file. (Particularly at this point, having a central Ansible variables file becomes invaluable, as typos and errors in these files cause a lot of headaches debugging an already complex distributed system like hadoop.)

```

- name: "Templating *-site.xml files for the node"
  template:
    src: "./Hadoop275/freebsd/{{item}}.j2"
    dest: "{{hdp_destdir}}/etc/{{hdp}}/{{item}}"
    owner: "{{hdp}}"
    group: "{{hdp}}"
  with_items:
    - core-site.xml
    - hdfs-site.xml
    - yarn-site.xml
    - mapred-site.xml

```

A file called `slaves` (newer versions renamed it to `workers`) contains the names of hosts that should serve as datanodes. The machine defined as the master can also participate and work on map-reduce jobs, hence the `localhost` in the file. The task here adds the workers that are defined as parameters to the Ansible playbook to that file:

```

- name: "Create and populate the slaves file"
  lineinfile:
    dest: "{{hdp_destdir}}/etc/{{hdp}}/slaves"
    owner: "{{hdp}}"
    group: "{{hdp}}"
    line: "{{item}}"
  with_items: "{{workers}}"

```

Now that a bunch of file changes have been made to the installation, we need to be sure that the files are still owned by hadoop and not the user running the Ansible script. This last task recursively sets ownership and group to the hadoop user on the files and directories the playbook that has touched so far.

```

- name: "Give ownership to {{hdp}}"
  file:
    path: "{{item}}"
    owner: "{{hdp}}"
    group: "{{hdp}}"
    recurse: yes
  with_items:
    - "{{hdp_home}}"
    - "{{hdp_destdir}}"
    - "/{{hdp}}"

```

That's the complete playbook called `freebsd_hadoop2.9.0.yml` and it can be executed with the following commandline.

```

$ ./freebsd_hadoop2.9.0.yml -Kbe 'host=namenode:datanode1:datanode2
master=namenode' -e '{"workers":["datanode1","datanode2"]}' --vault-id @prompt

```

The hosts `namenode`, `datanode1`, and `datanode2` all need to be defined in Ansible's inventory file. The `--vault-id` `@prompt` parameter will ask for the vault password that was defined when creating the vault.

Starting Hadoop and the First Map-reduce Job

After the playbook has been run and there are no errors in the deployment, it is time to log into the `namenode` host and switch to the `hadoop` user (using the password that was set). A first test is to verify that this user can log into each `datanode1` and `datanode2` without being prompted to confirm the hostkey or provide a password. If the login completes without any of these, then the hadoop services can be started. The first step is to format the distributed filesystem using the `hdfs namenode` command (the path to hadoop is in the `.bashrc` file, so the full path to the `hdfs` executable is omitted):

```
hadoop@namenode$ hdfs namenode -format
```

A couple of initialization messages scroll by, but there should be no errors at the end. Be careful when running this command a second time. Each time, a unique ID is generated to identify the HDFS from others. Unfortunately, the format is only done on the master node, not throughout the other cluster nodes. Hence, running it a second time will confuse the datanodes because they still retain the old ID. The solution is to wipe the directories defined in `hdp_data_dir` and `hdp_tmp` of any previous content, both on the datanodes and the namenode.

Next, all the services that make up the hadoop system must be started in order. The following commands will take care of that:

```
hadoop@namenode$ start-dfs.sh && start-yarn.sh && mr-jobhistory-daemon.sh start historyserver
```

To make sure all the processes have started successfully, run `jps` to verify that the following services have started on the namenode: `NameNode`, `ResourceManager`, `JobHistoryServer`, and `SecondaryNameNode`.

The datanodes must have these processes in the `jps` output: `NodeManager` and `DataNode`. (Running `jps` during a map-reduce job means more processes will be spawned on the datanodes that form the units of work the node is processing using the YARN framework.)

The cluster is ready to run its first map-reduce job. Hadoop provides sample jobs to get to know the framework without having to write a Java program first and packing it in a jar file to be executed as a job. One of these example files will try to calculate the value of pi using a Monte Carlo simulation. The following shell script can do that:

```
#!/bin/sh
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
hadoop jar /usr/local/hadoop2.9.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.0.jar pi 16 1000000
```

Executing the shell script will spawn mappers to calculate a subset of the Monte Carlo simulation. Depending on how many mappers are chosen (16 in this example), the accuracy of the result varies. The job can be monitored using a browser that's pointed to the URL `http://<the.namenode.ip.address>:8088`. Browsing to `http://<the.namenode.ip.address>:50070` displays the overall cluster status along with a filesystem browser for the HDFS and logs (manual refresh is required to get updated information on both pages).

Another interesting sample is the `random-text-writer` that creates a bunch of files in the HDFS across the nodes. A `timestamp` is used to make it possible to run this command multiple times in a row:

```

#!/bin/sh
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
timestamp=`date +%Y%m%d%H%M`
hadoop jar /usr/local/hadoop-2.9.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.0.jar randomtextwriter /random-text_$timestamp

```

After both jobs have run without errors, the hadoop cluster is ready to accept more sophisticated map-reduce jobs. This is left as a learning exercise for the reader, the internet is full of tutorials about how to write map-reduce jobs.

This tutorial closes with a view of the ZFS compression ratios achieved with the two jobs being completed (results may vary):

hadoop@namenode\$ zfs get refcompressratio hadooppool/hadoop/hdfs/namenode			
NAME	PROPERTY	VALUE	SOURCE
hadooppool/hadoop/hdfs/namenode	refcompressratio	26.28x	-

The datanodes also achieved quite a good compression ratio from the random-text-writer example:

NAME PROPERTY VALUE SOURCE			
hadooppool/hadoop/hdfs/datanode	refcompressratio	2.35x	-

This shows that running hadoop on FreeBSD has benefits. OpenZFS is able to add additional protection to the data stored in HDFS and makes it possible to store more data on the underlying disks. In a big data world, this is an enormous win.

BENEDICT REUSCHLING joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he actively began mentoring other people to become FreeBSD committers. He is a proctor for the BSD Certification Group and joined the FreeBSD Foundation in 2015, where he is currently serving as vice president. Benedict has a Master of Science degree in Computer Science and is teaching a UNIX for software developers class at the University of Applied Sciences, Darmstadt, Germany.

ZFS experts make their servers **ZING**

Now you can too. Get a copy of.....

Choose ebook, print, or combo. You'll learn to:

- Use boot environment, make the riskiest sysadmin tasks boring.
- Delegate filesystem privileges to users.
- Containerize ZFS datasets with jails.
- Quickly and efficiently replicate data between machines.
- Split layers off of mirrors.
- Optimize ZFS block storage.
- Handle large storage arrays.
- Select caching strategies to improve performance.
- Manage next-generation storage hardware.
- Identify and remove bottlenecks.
- Build screaming fast database storage.
- Dive deep into pools, metaslabs, and more!

Link to: <http://zfsbook.com>



WHETHER YOU MANAGE A SINGLE SMALL SERVER OR INTERNATIONAL DATACENTERS, SIMPLIFY YOUR STORAGE WITH **FREEBSD MASTERY: ADVANCED ZFS**. GET IT TODAY!



**Testers, Systems Administrators,
Authors, Advocates, and of course
Programmers** to join any of our diverse teams.

COME JOIN THE PROJECT THAT MAKES THE INTERNET GO!

★ DOWNLOAD OUR SOFTWARE ★

<http://www.freebsd.org/where.html>

★ JOIN OUR MAILING LISTS ★

<http://www.freebsd.org/community/mailingslists.html?>

★ ATTEND A CONFERENCE ★

BSDCam 2018 • August 15–17 • Cambridge, UK

EuroBSDCon 2018 • Sept 20-23 • Bucharest, Romania

MeetBSD 2018 • Oct 19–20 • Santa Clara, CA

The FreeBSD Project



pNFS

By Rick Macklem

A first attempt at creating a pNFS service for FreeBSD used GlusterFS along with the kernel-based "**nfssd**" communicating with it via fuse. It worked, but performance was abysmal with massive numbers of context switches. As such, I started over, designing an in-kernel service using the "**nfssd**" and some number of FreeBSD systems, but no cluster file system. This I refer to as **Plan B** and is discussed in this article.

Overall Goal

A pNFS service separates the Read/Write operations from all other NFSv4.1 operations. The hope is that this separation allows a pNFS service to be configured that exceeds the limits of a single NFS server for either storage capacity and/or I/O bandwidth. For a pNFS service, the NFS server becomes a Metadata Server (MDS) and handles all operations except for I/O operations. There are also other servers configured as data servers (DSs) that only handle I/O operations.

It is possible to configure mirroring within the DSs so that the data file for an MDS file will be mirrored on two or more of the DSs. When this is used, failure of a DS will not stop the pNFS service and a failed DS can be recovered once repaired while the pNFS service continues to operate. Although two-way mirroring would be the norm, it is possible to set a mirroring level of up to four or the number of DSs, whichever is less. The mirroring level refers to how many copies of the data file are kept on different DSs. The FreeBSD MDS is still a single point of failure, just as a normal NFS server is.

Overview of Plan B

A Plan B pNFS service consists of a single MDS and K DSs, all of which are FreeBSD12 systems. Clients will mount the MDS as they would a normal NFS server. When files are created, the MDS creates a file tree identical to what a normal NFS server creates, except that all the regular (VREG) files will be empty. As such, if you look at the exported tree on the MDS directly on the MDS server (not via an NFS mount), the files will all be of size 0. Each of these files will also have two extended attributes in the system attribute name space:

pnfsd.dsfile - This extended attribute stores the information that the MDS needs to find the data file(s) on DS(s) for this file.

pnfsd.dsattr - This extended attribute stores the Size, AccessTime, ModifyTime and Change attributes for the file, so that the MDS doesn't need to acquire the attributes from the DS for every **Getattr** operation.

For each regular (VREG) file, the MDS creates a data file on one or more, if mirroring is enabled, of the DSs in one of the "**dsN**" subdirectories. The name of this file is the file handle of the file on the MDS in hexadecimal so that the name is unique. The **ds(s)** are chosen in round-robin fashion when the file is created on the MDS and this seems to be adequate for storage of small files. For a service stor-

ing a mix of small and large files, a different algorithm may be needed.

I considered implementing a "most free space" algorithm, but have not done so, due to the overhead of checking how much free space each of the DSs has. The DSs use subdirectories named "`ds0`" to "`dsN`" so that no one directory gets too large. The value of "`N`" is set via the `sysctl vfs.nfsd.dsdirsize` on the MDS, with the default being 20.

For production servers that will store a lot of files, this value probably should be much larger. It can be increased when the "`nfsd`" daemon is not running on the MDS, once the additional "`dsN`" subdirectories are created on the DSs.

For pNFS aware `NFSv4.1` clients, the FreeBSD server will return two pieces of information to the client that allows it to do I/O directly to the DS.

- **DeviceInfo** - This is relatively static information that defines what a DS is. The critical information returned by the FreeBSD server is the IP address of the DS and, for the Flexible File layout, that it is "tightly coupled." There is a "`deviceid`" that identifies the DeviceInfo and which is used by the layout to reference it. The pNFS aware client acquires this information via the `NFSv4.1 GetDeviceInfo` operation.

- **Layout** - This is per file and can be recalled by the server when it is no longer valid. For the FreeBSD server, there is support for two types of layouts, called File and Flexible File layout respectively. Both allow the client to do I/O on the DS via `NFSv4.1` I/O operations. The Flexible File layout is a more recent variant that allows specification of mirrors, where the client is expected to do writes to all mirrors to maintain them in a consistent state. The Flexible File layout supports two variants referred to as "tightly coupled" vs "loosely coupled." The FreeBSD server always uses the "tightly coupled" variant, where the client uses the same credentials to do I/O on the DS as it would on the MDS. For the "loosely coupled" variant, the layout specifies a synthetic user/group that the client uses to do I/O on the DS. The FreeBSD server does not do striping and always returns layouts for the entire file. The critical information in a layout is Read vs Read/Write, the `deviceid(s)` that identify which DS(s) the data file is stored on and the file handle for the data file. The pNFS-aware client acquires this information via the `NFSv4.1 LayoutGet` operation. The client can also do a `LayoutReturn` operation to return the layout, either when it is done with it or when requested to do so by the `NFSv4.1` server doing a `CBLayourRecall` callback to the client. For Flexible File layouts, the client can report I/O errors when doing I/O on a DS to the MDS in the `LayoutReturn` arguments.

The MDS generates File layouts to `NFSv4.1` clients that know how to do pNFS for the non-mirrored DS case, unless the `sysctl vfs.nfsd.default_flexfile` is set non-zero, in which case Flexible File layouts are generated.

The mirrored DS configuration always generates Flexible File layouts. For NFS clients that do not support `NFSv4.1` pNFS, all I/O operations are sent to the MDS. When the MDS receives an I/O RPC, it will do the RPC on the DS as a proxy.

If the DS is on the same machine as the MDS, the MDS/DS will do the RPC on the DS as a proxy and so on, until the machine runs out of some resource, such as session slots or mbufs. Therefore, a DS cannot be on the same system as the MDS.

Although I wouldn't consider it a practical production setup, for testing you can use a single system for all DSs and that system can also be used as the client, allowing testing using only two systems. For testing, it is possible to create more than one DS on the DS system, but you must assign an alias IP address to this DS system for each additional DS and mount the additional DSs using these separate alias addresses. In other words, only one DS mount per IP address is allowed. The mounts must also use separate exported directories for each DS in which to store data files.

The pNFS service is in FreeBSD-current and will be in FreeBSD12 when it is released. Prior to the FreeBSD12 release, it is possible to use a FreeBSD-current snapshot distribution for testing.

Setting up a FreeBSD pNFS Server Using Plan B

Let's do an example assuming five FreeBSD12 systems, with four of them configured as DSs, using two-way mirroring and `AUTH_SYS`.

- The MDS, exporting `/export` to the clients.

`nfsv4-server`



- The DSs with `/DSstore` exported to the MDS and clients for storage of data files.
`nfsv4-data0, nfsv4-data1, nfsv4-data2` and `nfsv4-data3`

On `nfsv4-server`, you will need to export a file system tree for the clients. The two lines in `/etc/exports` might look like:

```
V4: /export -sec=sys -network 192.168.1.0 -mask 255.255.255.0
/export -sec=sys -network 192.168.1.0 -mask 255.255.255.0
```

Then you will need the following lines in your `/etc/rc.conf`:

```
rpcbind_enable="YES"
mountd_enable="YES"
nfs_server_enable="YES"
nfsv4_server_enable="YES"
nfs_server_flags="-u -t -n 32 -p nfsv4-data0,nfsv4-data1,nfsv4-data2,
nfsv4-data3 -m 2"
```

Unless you wish to run the `nfsuserd` to map between uid/gid numbers and names, put these lines in `/etc/sysctl.conf`:

```
vfs.nfs.enable_uidtostring=1
vfs.nfsd.enable_stringtouid=1
```

This configures the NFSv4.1 server to use uid/gid numbers in the `owner` and `owner_group` strings.

Then the DSs must be mounted on the MDS. `/etc/fstab` lines like:

<code>nfsv4-data0:/</code>	<code>/data0</code>	<code>nfsrw,nfsv4,minorversion=1,soft,retrans=2</code>	<code>0</code>	<code>0</code>
<code>nfsv4-data1:/</code>	<code>/data1</code>	<code>nfsrw,nfsv4,minorversion=1,soft,retrans=2</code>	<code>0</code>	<code>0</code>
<code>nfsv4-data2:/</code>	<code>/data2</code>	<code>nfsrw,nfsv4,minorversion=1,soft,retrans=2</code>	<code>0</code>	<code>0</code>
<code>nfsv4-data3:/</code>	<code>/data3</code>	<code>nfsrw,nfsv4,minorversion=1,soft,retrans=2</code>	<code>0</code>	<code>0</code>

and the `/data0`, `/data1`, `/data2` and `/data3` directories will need to be created on the MDS for mounting of the DS data files. Note that "`soft,retrans=2`" would not normally be used for an NFSv4 mount, but this is an exception, since no state operations are done on the DS. Doing this allows the proxy operations to a DS to fail and cause the failing DS to be disabled when this occurs.

On the DSs, you will need to `mkdir` and export the `/DSstore` directory to the MDS and clients. The `/etc/exports` lines might look like:

```
V4: /DSstore -sec=sys -network 192.168.1.0 -mask 255.255.255.0
/DSstore -sec=sys -maproot=root nfsv4-server
/DSstore -sec=sys -network 192.168.1.0 -mask 255.255.255.0
```

You will need the following lines in your `/etc/rc.conf`:

```
rpcbind_enable="YES"
mountd_enable="YES"
nfs_server_enable="YES"
nfsv4_server_enable="YES"
nfs_server_flags="-u -t -n 32"
```

And in `/etc/sysctl.conf`:

```
vfs.nfs.enable_uidtostring=1
vfs.nfsd.enable_stringtouid=1
```

You will need to create the "dsN" directories under `/DSstore`. This command done in `/DSstore` on each of the DSs will do it: (All commands from here on will need to be done by `root/su`.)

```
# jot -w ds 20 0 | xargs mkdir -m 700
```

Once these systems are set up, the MDS should be ready for client mounts. The FreeBSD clients will need the following in their `/etc/rc.conf` files:

```
rpcbind_enable="YES"  
nfs_client_enable="YES"  
nfscbd_enable="YES"
```

Then, on the FreeBSD client, the mount command might look like:

```
# mount -t nfs -o nfsv4,minorversion=1,pnfs nfsv4-server:/ /mnt
```

The client can then use "/mnt" as it would a normal NFS mount. If you do "nfsstat -E -s" on **nfsv4-server**, you should not see very many Read or Write operations. Most Read and Write operations should show up on a "nfsstat -E -s" done on the DSs. Having a few Read and Write operations on the MDS is normal, since that is what the clients fall back on when they fail to get a valid layout for any reason.

If, instead, you did a NFSv3 mount, the command might be:

```
# mount -t nfs nfsv4-server:/ /mnt
```

Now, if you do "nfsstat -E -s" on **nfsv4-server**, you will see a lot of Read and Write operations, since they are all being done through the MDS, which acts as a proxy for the DSs.

Let's suppose you create a file called "abc.c" on /mnt that is 274 bytes in size. Doing "ls -l" in /mnt would show a line like:

```
-rw-r--r--    1 rickst wheel  274 Jun  5 18:02 abc.c
```

Whereas if you go to `/export` on `nfsv4-server`, the `"ls -1"` line will look like:

```
-rw-r--r-- 1 ricktst  wheel  0 Jun  5 18:02 abc.c
```

Then, a "lsetxattr system abc.c" in the same directory will show:

abc.c pnfsd.dsfile pnfsd.dsattr

and a "pnfsdssfile abc.c" will show:

abc.c: nfsv4-data2

(The `pnfdsdsfile` command shows what is in "pnfsd.dsfile" unless it has command line arguments specified.)

This tells you that the data for "abc.c" is stored on `nfsv4-data2` and `nfsv4-data3` in subdirectory "ds5" with filename "2075...".

If we go to /DSstore/ds5 on nfsv4-data2, an "ls -l *a97302*" will show:

and on nfsv4-data3:

```
-rw-r--r-- 1 ricktst  wheel  274 Jun  5 18:02
207508569ff983350c000000a9730200eec58e800000000000000000000000000000000
```

Note that the ownership and permissions are the same as "abc.c" on the MDS. This is because it is a "tightly coupled" Flexible File layout service and enforces permission checking on the DS. For the Flexible File layout, the client can write to both **nfsv4-data2** and **nfsv4-data3** concurrently, so there should not be a significant performance hit caused by the mirroring. However, twice as much storage will be used as a non-mir-

rored configuration would use. The "atime" is not normally kept consistent across the DSs, but if the sysctl vfs.nfsd.pnfsstrictatime is set to one, it will be. Setting this does result in significant overheads.

Ok, so now let's have some fun with it. The MDS will disable a mirrored DS when one of three things occurs:

1. The MetaData Server (MDS) detects a problem when trying to do a proxy operation on the DS. This is why the DS servers are mounted on the MDS with the "`soft, retrans=2`" options. This can take a couple of minutes after the DS failure or network partitioning occurs.

2. A pNFS client can report an I/O error with respect to a DS to the MDS in the arguments for a LayoutReturn operation.

3. The system administrator can perform the `pnfsdskill(1)` command on the MDS to disable a DS. If the system administrator does a `pnfsdskill(1)` and it fails with `ENXIO` (Device not configured) that normally means the DS was already disabled via #1 or #2. Since doing this is harmless, once a system administrator knows that there is a problem with a mirrored DS, doing the command is recommended.

Let's do #3, since it is easy:

```
# pnfsdskill /data2
```

This will log a message on the console:
pNFS server: mirror **nfsv4-data2** failed

`nfsv4-data2` has now been marked disabled and `CBLLayoutRecall` callbacks have been done for all layouts using `nfsv4-data2`.

We can then unmount the `nfsv4-data2 /DSstore`:

```
# umount -N /data2
```

(The `-N` option ensures that the `umount` works even if threads are stuck trying to do RPCs on a failed `nfsv4-data2`.)

Now, let's write some data into `/mnt/abc.c`, so an `"ls -l"` line on the client looks like:

```
-rw-r--r-- 1 ricktst  wheel  586 Jun  5 19:11 abc.c
```

-rw-r--r-- 1 rickst wheel 274 Jun 5 18:02 .vmlinux

whereas on `nfsv4-data3`:

Notice that `nfsv4-data2` didn't get written, since it is disabled. The file "`abc.c`" can still be read/written, but there is no longer a redundant copy.

The first step in repairing **nfsv4-data2** is to make sure that the out-of-date copy of the file on **nfsv4-data2** doesn't get used when **nfsv4-data2** is brought back online. To do this, we go to the **/export** directory on the MDS and replace **nfsv4-data2** with IP address **0.0.0.0** via the command:

so, now it won't try to use `nfsv4-data2`.

This has to be done for all files in `/export` that specifies `nfsv4-data2` as a DS, so using `find(1)` the command is:

```
# find . -type f -exec pnfsdsfile -q -r nfsv4-data2 {} \;
```

Note that many files won't have `nfsv4-data2` specified as a DS, but the command knows to just `exit(0)` for these, so it can be safely used on any file.

Unfortunately, some combination of "rename" or "link/unlink" can result in `find(1)` missing some files. To check for missing ones, the command is:

```
# find . -type f -exec pnfsdsfile {} \; | sed "/nfsv4-data2/!d" | sed "s/:.*//"
```

(Using the "sh" shell to search for any files still specifying `nfsv4-data2`.)

The "`pnfsdsfile -r`" command needs to be done for any file names printed out by the above.

So, now we can safely bring `nfsv4-data2` back online after fixing it. To fix it for this exercise, we'll just go into `/DSstore` on `nfsv4-data2` and clean it up:

```
# cd /DSstore
# rm -rf *
# jot -w ds 20 0 | xargs mkdir -m 700
```

Now, on the MDS, we can bring it back online by mounting it and restarting the `nfsd` daemon:

```
# mount -t nfs -o nfsv4,minorversion=1,soft,retrans=2 nfsv4-data2:/ /data2
# /etc/rc.d/nfsd restart
```

After doing the above, newly created files may be assigned to `nfsv4-data2`, but the ones previously mirrored on `nfsv4-data2` still need to be recovered. To do this, we go to `/export` on the MDS and do the command:

```
# pnfsdscopymr -r /data2 abc.c
which copies the data for abc.c onto nfsv4-data2.
```

After doing this command, `pnfsdsfile(1)` shows:

```
# pnfsdsfile abc.c
abc.c: nfsv4-data2.home.rick
ds5/207508569ff983350c00000a9730200eec58e80000000000000000000000
nfsv4-data3.home.rick
ds5/207508569ff983350c00000a9730200eec58e80000000000000000000000
```

If we go to `/DSstore/ds5` in `nfsv4-data2`, an "`ls -l *a97302*`" will show:

```
-rw-r--r-- 1 ricktst wheel 586 Jun 5 19:11
207508569ff983350c00000a9730200eec58e80000000000000000000000
```

and on `nfsv4-data3`:

```
-rw-r--r-- 1 ricktst wheel 586 Jun 5 19:11
207508569ff983350c00000a9730200eec58e80000000000000000000000
```

The code that implements this copy in the kernel is somewhat involved. A brief description of the algorithm is:

- The MDS file's vnode is locked, blocking `LayoutGet` operations.
- Disable issuing of `Read/Write` layouts for the file via the `nfsdontlist`, so that they will be disabled after the MDS file's vnode is unlocked.
- Set up the `nfsrv_recalllist` so that recall of `read/write` layouts can be done.
- Unlock the MDS file's vnode, so that the `client(s)` can perform proxied writes, `LayoutCommits` and `LayoutReturns` for the file when completing the `LayoutReturn` requested by the `LayoutRecall` callback.
- Issue a `CBLLayoutRecall` callback for all `Read/Write` layouts and wait for them to be returned. (If the `CBLLayoutRecall` callback replies `NFSERR_NOMATCHLAYOUT`, they are gone and no `LayoutReturn` is needed.)
- Exclusively lock the MDS file's vnode. This ensures that no proxied writes are in progress or can occur during the DS file copy.

It also blocks `setattr` operations.

- Create the file on the repaired mirror.
- Copy the file from the operational DS.
- Copy any ACL from the MDS file to the new DS file.
- Set the modify time of the new DS file to that of the MDS file.
- Update the extended attribute for the MDS file.
- Enable issuing of `Read/Write` layouts by deleting the `nfsdontlist` entry.
- Unlock the MDS file's vnode allowing operations to continue normally, since it is again mirrored.

Again, this has to be done for all files, so the `find(1)` command is:

```
# find . -type f -exec pnfsdscopymr -r /data2 {} \;
```

and to check for ones missed by the `find(1)`:

```
# find . -type f -exec pnfsdsfile {} \; | sed "/0\.\0\.\0\.\0/!d" | sed("/:.*//")  
(Using "sh", search for any files that still have a DS address of 0.0.0.0.)
```

If this prints `file(s)`, the "`pnfsdscopymr -r`" command needs to be done on them. If nothing gets printed out, `nfsv4-data2` has been recovered and all files should now be mirrored correctly.

A system administrator can also use the `pnfsdscopymr(1)` command to migrate the data file from one DS to another DS. To move the data file for `abc.c` from `nfsv4-data3` to `nfsv4-data0`, the command is:

```
# pnfstdscopymr -m /data3 /data0 abc.c
```

After this command, `pnfsdsfile` shows:

```
# pnfsdsfile abc.c
abc.c:      nfsv4-data2.home.rick
ds5/207508569ff983350c000000a9730200eec58e80000000000000000000000
nfsv4-data0.home.rick
ds5/207508569ff983350c000000a9730200eec58e80000000000000000000000
```

If we go to /dsstore/ds5 in nfsv4-data2, an "ls -l *a97302*" will show:

```
on nfsv4-data3:  
ls: No match
```

and on nfsv4-data0:

This might be useful to move the data for a large file to a DS with more free space.

The "pnfsdscopymr -m" command just does some sanity checking followed by one system call to do the work. This system call could be used by a storage/load balancer implementation in the future.

The Linux client currently has a Flexible File layout driver that supports the "loosely coupled" variant but does not handle the "tightly coupled" variant correctly. It always uses the synthetic user/group for I/O operations on a DS.

There are two ways to deal with this:

1. Patch the client driver to fix this. I have a patch here:
<http://people.freebsd.org/~rmacklem/flexfile.patch> which seems to work ok.
2. Export `/DSstore` on the DSs with `"-maproot=root"` and then set the `vfs.nfsd.flexlinuxhack=1` on the MDS. Setting this sysctl makes the pNFS server send the synthetic/group of `(0, 0)` in the layout, so that the Linux client always does I/O on the DSs as `"root"`.

You also need a recent Linux kernel. I have been testing with Linux-4.17-rc2. Linux-4.12 worked but would crash intermittently during testing and earlier kernels don't have Flexible File layout support for NFSv4.1 DSs. This is not a problem for the non-mirrored pNFS server, since it will send File layouts to the Linux client.

Once you have resolved this, the mount command on Linux is:

```
# mount -t nfs -o nfsvers=4,minorversion=1 nfs4-server:/ /mnt
```

For more information on the setup and management of the pNFS service, the document <http://people.freebsd.org/~rmacklem/pnfs-planb-setup.txt> might be useful. There are also man pages for **pnfsdskill(1)**, **pnfsdssfile(1)**, **pnfsdscopymr(1)** and **pnfs(4)**.

If you wish to look at the protocol details, the RFCs are:

RFC-5661: Network File System (NFS) Version 4 Minor Version 1 Protocol, ISSN: 2070-1721.

The flexible file layout is currently an internet draft but should be published as an RFC soon. Hopefully before this article is published. If not, here is the draft:

Parallel NFS (pNFS) Flexible File Layout [draft-ietf-nfsv4-flex-files-19.txt](http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-flex-files-19.txt).



Rick Macklem has been working with BSD for way too long. His first contribution to BSD was a port of 4.2BSD to the MicroVAXII in 1985. Shortly after that, he contributed the first NFS implementation that became part of 4.3BSD Reno. He worked as a sysadmin for a Canadian university for thirty years and is now happily retired and still working on NFS for FreeBSD. And, yes, this article was written using "ed", which is still his editor of choice.



FreeBSD®

The FreeBSD Project is looking for

- Programmers
- Testers
- Researchers
- Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website
freebsd.org/projects/newbies.html

Downloading the Software
freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at
freebsdfoundation.org

Help Create the Future. Join the FreeBSD Project!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by





High-Performance Computing & FreeBSD

By Johannes M. Dieterich

High-performance computing encompasses a variety of fields and domains, from science and engineering to finance and social studies. The common denominator is *the practice of aggregating computing power in a way that delivers much higher performance than one could get from a typical desktop computer or workstation to solve large problems* [1]. More concrete examples include, in order of descending compute share on civil installations: materials design, drug discovery, climate modeling, materials design, computational fluid dynamics, economic simulations, and big data analysis.

Arguably, we are living in a consolidated HPC world in terms of operating system and microarchitectures employed, and on a cursory glance, contemporary HPC installations look very much alike. A typical installation will feature a Beowulf cluster build from thousands of nodes connected by a fast interconnect. As of this writing, the TOP500 list of the fastest supercomputers in the world contains 91.5% amd64 processors and 99.6% Linux operating systems (OSs) [3].

So where is the space for FreeBSD in this monoculture and why should the FreeBSD community care about HPC? First, HPC continues to be both

the source of important basic technologies widely used in more general computing and a cauldron of innovative technologies, e.g., basic numerical libraries and more recently deep-learning applications. Second, a more in-depth view would reveal a much more complex ecosystem also containing communication and the aforementioned numerical libraries, toolchains, and programming languages, as well as auxiliary hardware solutions like network switches and storage. Lastly, there are plenty of systems, such as developer workstations, that are more diverse, have different capability requirements, and are more accessible to OS alternatives; e.g., Ubuntu Linux is a prime choice for developer workstations, not so much for cluster installations.

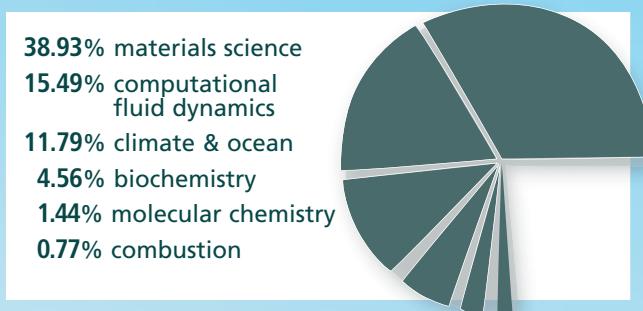
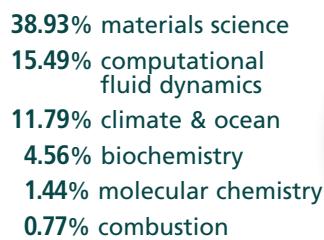


Fig.1 FLOP usage by domain in HPC on a typical supercomputer. Ref.[2]

To some extent, one may even argue that cloud computing is repackaged or broken-down HPC for customers not traditionally having or being able to afford a stake in or access to large installations. Even though FreeBSD already excels as an appliance solution, other capabilities (hypervisor choice, storage, security, network, etc.) matter more than the compute OS. I will focus here on capabilities for HPC, either the compute or developer OS, how they map onto more general computing needs, and try to highlight how FreeBSD would benefit from some HPC-targeted improvements.

As FreeBSD users and developers, we are aware of and value the inherent advantages of FreeBSD: a consistent experience of applications and libraries in the ports system, the easy build and deployment of custom packages potentially with architecture-specific optimizations, and, in general, a very stable operating system experience. Most importantly, the harmonized ports experience allows us to propagate changes across all parts of the OS interacting with the user or developer consistently.

Languages

What are the tools of HPC—the applications used by researchers to transform FLOPS into knowledge—made from? If we remember that HPC was one of the original purposes of computers, the answer that the majority of computing time is used by Fortran codes is less of a surprise. More than 65% of compute cycles are spent on Fortran codes on a typical installation. This share increases if one considers the numerical Fortran libraries used in mixed-language codes. Unfortunately, this shows that Fortran is alive and unlikely to go anywhere in the near-term to mid-term due to the size and complexity of existing code bases. Hence, HPC support absolutely requires Fortran support. Due to the effective absence of the typically used commercial compilers on FreeBSD, we fall back to the open-source alternatives.

Currently, if a port specifies `USES=fortran` all architectures will use the `gfortran` compiler from `lang/gcc`. `gfortran` is a good Fortran compiler that produces stable, fast binaries and supports all relevant Fortran standards. Currently, using `gfortran` requires explicit specification of the `rpath` to the linker to search for GNU libraries for both that port and, if it is a library, all dependent ports (or out-of-ports tree applications, for that matter). While trivial, it significantly increases the maintenance burden of porters and developers. Remedies are being discussed currently, but no patches have landed in the tree yet. Also recall that

all possible architectures (among them `amd64`) use the LLVM-based `clang` compiler as the base compiler for licensing reasons and, hence, by extension, as the default ports compiler. Instead of sorting out mixed-compiler environments, it is then quite often easier to simply rely fully on the GNU compilers for mixed-language programs. This is a mildly unsatisfying situation.

Is there a more fundamental improvement possible? Maybe. `flang` is an open, LLVM-based, new Fortran compiler. [4] Its frontend is derived from the well-established Portland Group's compiler and was open-sourced and continues to be maintained by NVIDIA. It since has found support and use by AMD in its AOCC package as well. `flang` supports only up to Fortran 2003 language features and is limited to 64-bit architectures. FreeBSD now contains `devel/flang` as a preview. Certainly, some time and effort are required to make this port competitive with `gfortran` and vet it properly for HPC uses. Just very recently, plans from NVIDIA became public to rewrite large parts of `flang` to improve its feature set and likelihood of being accepted under the official LLVM umbrella. Uptake within the HPC community will be interesting to observe, but the added competition should prove beneficial to the GNU compiler either way.

Let us think beyond the single core. As noted above, a typical cluster contains tens of thousands of cores, and jobs are typically required to use tens to hundreds of them in parallel. Two major approaches exist for scaling out in HPC: OpenMP [5] and the Message Passing Interface (MPI) [6].

OpenMP mostly targets shared-memory machines with later standards having support for accelerator offloading and is discussed below. It is a relatively simple, pragma-based approach, supports C/C++ and Fortran, and is commonly used to parallelize over loops in a data-parallel fashion. Currently, our ports tree will again default to the `lang/gcc` port if a `USES= compiler:openmp` is encountered. For this reason, ports typically will not enable OpenMP by default (including some commonly used ones like `graphics/ImageMagick`) and hence will be limited to a single core.

A better alternative exists for at least `amd64` and `i386`; the `libomp` library associated with the LLVM project since the initial open-source release by Intel. It has not been imported into base yet, but a review for an older library version exists. In its hopefully temporary absence, the ports system should use one of the `devel/llvm` ports that do include `libomp`. Multiple integration tests have been done and the feature should soon be ready to land. Hopefully this will also incentivize fellow developers

to add the necessary FreeBSD bits for other architectures supported upstream. My tests indicate that `libomp` integration in LLVM is not ideal from a performance perspective; in particular, LLVM's vectorizer does not work (well) if OpenMP's `simd` pragma is used in conjunction with standard OpenMP thread parallelization (e.g., `parallel for`). But certainly, suboptimal parallelization is preferable to no parallelization.

MPI on the other hand operates through function calls into the MPI communication library. It features both simple send/receive/broadcast features as well as more advanced operations like reductions. MPI is used both for process-based intra-node, as well as inter-node, parallelization, and on the hardware level, typically uses a fast interconnect such as InfiniBand. Typically, MPI-enabled software packages are compiled using `mpicc/mpif90` wrapper scripts which configure the underlying compilers to find MPI headers and link against the MPI library. Within the ports tree, multiple MPI choices exist and we are only limited by the underlying compiler toolchain for Fortran.

Numerical Libraries

A large part of the *high performance* in HPC does not result from application code, but instead, in the judicious and abundant use of highly-optimized numerical libraries implementing standardized APIs. The arguably most important APIs are BLAS, a collection of basic dense linear algebra operations such as matrix-matrix multiplications, LAPACK, a collection of more complicated solvers such as Cholesky or LU decompositions, and Fast Fourier Transformations (FFTs), typically in the FFTW3 API incarnation. Their fundamental nature also makes them a common dependency in our ports system, e.g., for audio and graphics applications.

Choice in libraries implementing these APIs is much less important than performance and features of a single set of them. Most HPC clusters

provide vendor-tuned, assembly-optimized BLAS and LAPACK libraries. For FreeBSD, we have multiple options exposed through the `blaslapack` selector. By default, this selector will use `math/blas` and `math/lapack`. These are Fortran-source, reference implementations and, as such, not competitive with optimized alternatives such as `math/blis` and `math/libflame`. Both the reference implementations and `math/openblas` have a Fortran dependency unlike the FLAME project's `BLIS` and `libflame` [7].

As we can see from Figure 3, both the OpenBLAS or BLIS implementations show a commanding lead in performance, starting from small to medium-sized matrix-matrix multiplications if their CPU architecture optimized kernels are used. More importantly, even if BLIS's source-only reference implementation is used, the implementation and blocking scheme results in an up to 80% performance advantage. Additionally, BLIS provides the ability to use `pthread` parallelization, which is of less impact in this test case. The FLAME project has expressed interest in working with us, accepting pull requests of FreeBSD changes and implementing features such as runtime kernel selection which we need for generic packages. Hence, BLIS is a good candidate to be our default BLAS implementation and rids us of a low-level Fortran dependency.

The `math/libflame` port has recently been updated to a recent development snapshot and configured to expose a LAPACK interface for `amd64` and `i386` CURRENT. More vetting is required before the FLAME libraries can be added as a `blaslapack` option for recent releases. Work in this regard is ongoing.

The status of other numerical, engineering, and scientific libraries on FreeBSD is already excellent: the `math/fftw3` port is in great shape; we also have a variety of development libraries from a range of domains including quantum physics/chemistry ready to use and multiple active ports committers in that realm.

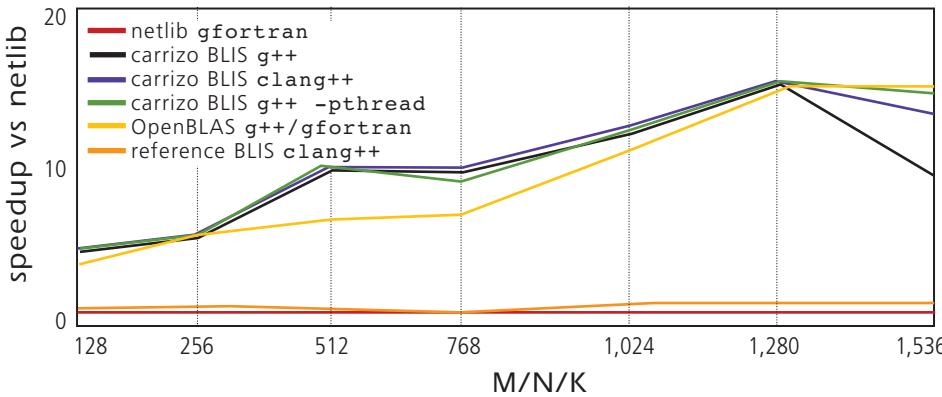


Fig. 3. Speedup of BLIS and OpenBLAS over the reference netlib BLAS for matrix-matrix multiplications (`dgemm`) of different sizes $M/N/K$ on an AMD A12-8800B CPU. Data averaged over 1,000 evaluations on FreeBSD HEAD, compilation with `-O2`, all libraries compiled with `-mavx -mavx2 -msse -msse2 -msse3 -msse4 -msse4a -msse4.1 -msse4.2 -mmmx -maes -mbmi -mbmi2 -mf16c -mfsgsbase -mtune=bdver4`, BLIS compiled with `clang++ 5.0.0`, netlib/OpenBLAS with `g++ 6.4.0`.

Developing on and Porting to FreeBSD

With an overall, relatively positive status of the languages and libraries on FreeBSD, how hard is porting HPC applications to and developing them on FreeBSD? Typical challenges arise from the use of nonstandard make systems (typically combinations of shell and other scripts as well as make files), the prevalent use of Linuxisms [8] such as hardocded system paths and other assumptions, the generally nonstandard compliance of code (e.g., it only compiles and yields the correct result with a specific proprietary compiler [version]), and also our own BSD-isms such as the `rpath`. All but the last challenge are arguments in favor of porting: code bases typically improve and lingering bugs are exposed and fixed. Porting to FreeBSD has tenable advantages if done properly: just as with any other application, changes should be upstreamed, explained, and continuously maintained. In the meantime, we should consider reducing our BSD-isms to ease the porting task.

Continuous maintenance or development on FreeBSD is straightforward. Even though the normal commercial toolchains for profiling, debugging, etc., are absent, the base system and ports collec-

tion includes excellent free alternatives. `dtrace` and `hwpmc` in conjunction with `benchmarks`/`flamegraph` make analyzing application performance from the user down to the kernel level straightforward. `devel/gdb` and `lldb` may not have the same level of graphical user interface support as the commercial alternatives but do get the job done. Even though most HPC code is still developed with `vi` or `emacs`, modern editor/IDE alternatives are present with `java/eclipse`, `java/netbeans`, and the `linuxulator`-using `editors/linux-sublime3`.

Recently, I have found `bhyve` provides an invaluable addition to the HPC developer's toolkit. No matter whether bugs are to be located only occurring on a particular platform configuration, specific Linux releases need support, or a full, continuous integration setup is needed, `bhyve` provides an excellent solution for all these use cases.

Accelerators

Probably the most disruptive change to the HPC landscape in the last decade was the introduction of accelerators into the HPC mainstream and its subsequent trickling into the workstation and mainstream market. Even though most TOP500 installa-

Thank you!

The FreeBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.



Are you a fan of FreeBSD? Help us give back to the Project and donate today! freebsdfoundation.org/donate/

Please check out the full list of generous community investors at freebsdfoundation.org/donate/sponsors

Iridium



NetApp®

Silver



vmware



tions still do not contain accelerators, the importance of accelerators for HPC and workstation applications cannot be overstated. Various means to exploit their potential exist, with the most important ones being direct programming via NVIDIA's proprietary CUDA language, the open alternative OpenCL, or through offloading via OpenMP.

These programming frameworks all rely on three fundamental parts: kernel driver support, a compiler or library, and a runtime environment. Through the FreeBSDDesktop project, we now have access to more recent open drivers for AMD and Intel GPUs. NVIDIA GPUs continue to be supported by the binary driver in the ports. There is no official CUDA support for FreeBSD; however, some people have reported success in the past compiling CUDA applications on Linux and running them through the Linuxulator on FreeBSD. Our OpenCL support is in reasonable shape: we include the nonofficial OpenCL clover library from the Mesa project for AMD GPUs. Its performance is absolutely not competitive but it works relatively reliably. For Intel, we include their official beignet implementation; however, Intel's GPUs are less competitive for compute tasks. Developing OpenCL applications is well supported; we include the CPU OpenCL emulator `lang/pocl` and the sanity checker `devel/oclgrind`. The exploitation of accelerators through OpenMP offloading is not supported currently but is, in general, not widespread yet.

A major improvement could be to include the Radeon Open Compute (ROCm) project [9]. It needs an open companion kernel driver, `amdkfd`, for the regular open `amdgpu` driver and provides a large open ecosystem centered around LLVM compiler technology to support both OpenCL and an open competitor to CUDA called HIP on AMD's GPUs. The FreeBSDDesktop team is now actively working on porting `amdkfd`, and the large ROCm stack should be straightforward if somewhat labor-intensive to port.

What's Next?

The highest priorities should be to vet FLAME's BLAS and LAPACK libraries and add them as an option to the `blaslapack` selector. Secondly, `libomp` should be used as the default for `amd64` through `devel/llvm`. Subsequently, some stabilization and extension of these major changes to architectures other than `amd64` will be needed. In the mid-term, I am hoping that we will be able to have ROCm working on FreeBSD. Another big-ticket item is proper SIMD/vectorization support in

our `libm` and from LLVM. Together, these should already be an interesting HPC platform for developers. Hopefully, in the long-term, we can improve the Fortran situation and make FreeBSD a truly compelling HPC alternative.

It is also important to realize that improving FreeBSD for HPC will not hurt it, either as a server or workstation system. On the contrary, it will likely be a boon for these use cases.

Acknowledgments

I wish to express my gratitude to all FreeBSD developers and users for making FreeBSD the platform it is. In particular, I would like to thank the FreeBSDDesktop team and my mentors Matthew Macy, Niclas Zeising, Steve Wills, and Rene Ladan. I also extend my deep gratitude to the BSDTW conference and organizers where the content of this article was first presented as a lecture. ●

REFERENCE LIST

- [1] InsideHPC: <https://insidehpc.com/hpc-basic-training/what-is-hpc/>
- [2] Data source: UK supercomputer ARCHER application usage of the last month, <http://www.archer.ac.uk/status/codes>
- [3] Data source: TOP500 list, June 2017, <https://www.top500.org/statistics/list/>
- [4] Flang github: <https://github.com/flang-compiler/flang>
- [5] OpenMP <https://www.openmp.org/>
- [6] MPI forum <https://www.mpi-forum.org/>
- [7] FLAME project <https://github.com/flame>
- [8] Linuxisms discussion <https://wiki.freebsd.org/AvoidingLinuxisms>
- [9] Radeon Open Compute <https://github.com/RadeonOpenCompute/>

JOHANNES DIETERICH started using FreeBSD with the 6.1 release and became a ports committer a year ago. During the day, he has spent the last nine years in academic research working on high-performance computing for a range of problems from global optimization to quantum chemical methods. Recently, he started working on GPU-accelerated deep learning for AMD.

BSD CERTIFICATION GROUP HAS JOINED WITH LPI.



ONE MORE STEP TOWARDS A
FREE ^{AND} OPEN
SOURCE WORLD

NOT TO MENTION, JOBS!

Now as a part of LPI, BSD certification will gain a new global reach. It will also benefit as it's relaunched in 2018 to fit into a broader program of free and open source professional credentials. You can help by participating in retooling the certification at lpi.org/bsd.



**COMING IN 2018.
WATCH FOR UPDATES.**



FreeBSD and **NVM EXPRESS**

By Jim **Harris** and Warner **Losh**

NVM Express (NVMe) has quickly become the predominant standard for high-performance, non-volatile memory access across PCI Express. FreeBSD added support for NVMe in 2012, enabling FreeBSD to take advantage of devices that can deliver over 500,000 IO/s per NVMe device¹. Companies like Netflix have rapidly moved to deploying NVMe storage on FreeBSD, such that FreeBSD's NVMe subsystem is now helping drive a significant portion of North American internet traffic². In this article, we describe the NVMe specification and FreeBSD's implementation of the specification and provide an overview of FreeBSD's utilities for monitoring and managing NVMe storage.

Let's first describe some of the common terms used in NVMe. An NVMe SSD is referred to as an NVMe controller and is roughly equivalent to a SCSI HBA or an AHCI controller. The primary difference is that for NVMe, the media resides within the controller itself – there is no separate protocol nor cabling between the controller and its media. The storage media within the NVMe controller is grouped into one or more NVMe namespaces. A namespace can be considered roughly equivalent to a SCSI LUN. This combination of the HBA and media into one unit reduces the amount of overhead by simplifying the protocol needed and eliminating layers of abstraction. The design uses the capabilities of PCIe to eliminate driver bottlenecks by supporting lockless queueing of requests.

In FreeBSD, NVMe controllers and namespaces are enumerated and initialized through the `nvme(4)` driver. `nvme(4)` is also responsible for providing an interface to expose these namespaces as block devices, but does not register those namespaces with GEOM nor CAM directly. `nvd(4)` registers each namespace with GEOM as a block device. Since NVMe has become mainstream since `nvd(4)` was originally developed, Netflix added `nda(4)` as an alternative to `nvd(4)`. The `nvd(4)` driver is a very thin layer on top of the NVMe protocol and is designed to operate at high transaction rates. The `nda(4)` driver integrates with CAM, including its error recovery and advanced queueing. It also offers traffic shaping to the drive via the I/O scheduler to improve overall performance.

NVMe System Integration

As you can see from this idealized picture, `nvd(4)` integrates directly to the disk layer. This has the advantage of queueing the commands to the NVMe drive with as little delay as possible. Since NVMe drives are designed to scale, this works out fairly well. The minimal delay in getting I/O to the drive translates to a lower latency and simpler code path.

In contrast, the `nda(4)` driver connects through CAM. CAM schedules the I/O to the

	User Space
	System Calls
	File System
	Buffer Cache
	GEOM
	Disk Interface
<code>nvd</code>	<code>nda</code>
	CAM
	<code>nvme_sim</code>

drive. This can involve reordering commands at times. The CAM system is designed to accept a high queue rate as to have the queued I/O sent to the drive as the I/O is available for it. CAM also adds error handling which can help recover drives when things go wrong better than `nvd(4)` (which has little error recovery in its error path). `nda(4)` can also shape I/O traffic to the drive, to a limited extent, through the CAM I/O scheduler. This can be utilized to introduce some unfairness to produce better results. For example, video streaming benefits from a bias towards read operations. Many NVMe drives have trouble with TRIMs but will soon be able to heavily throttle or shape the trims to the drive. `nda(4)` is able to collapse multiple trims into one trim request for the drive.

The `nvme(4)` driver is responsible for enumerating the nvme drives and their namespaces when the driver is attached. The `nvd(4)` or `nda(4)` drivers register interest in these (and other) events with the `nvme_register_consumer()` call at startup. So, when it detects a new namespace, it calls the new namespace callback. For the `nvd(4)` driver, a new disk interface is created and registered with GEOM. With the `nda(4)` driver, the callback goes to `nvme_sim` which creates the appropriate CAM devices that cause `nda(4)` to be created (CAM's SCSI legacy means that it does many things via indirection or deferred callback). The `nda(4)` driver then creates the disk and registers it with GEOM.

Regardless of the disk interface, when requests come from the upper layers of the system (specifically from the disk layer via command structures called bios), the `nvd` or `nda` will convert the `BIO_READ`, `BIO_WRITE` and `BIO_DELETE` commands into the appropriate `nvme` command and pass those requests to the `nvme(4)` driver for execution.

Exposing NVMe namespaces via `nvd(4)` versus `nda(4)` is controlled by the `hw.nvme.use_nvd` tunable. It currently defaults to 1, meaning namespaces are exposed via `nvd(4)`.

NVMe Commands and Queue Pairs

NVMe commands such as `READ`, `WRITE` and `IDENTIFY` are submitted by the host to the controller via submission queues, with the controller notifying the host about completions of those commands via completion queues. The specification allows for multiple submission queues to share one completion queue, but FreeBSD always associates one submission queue with one completion queue to form a logical queue pair or "qpair". These qpair associations are critical for unlocking NVMe parallelism on many-core systems and will be described later in this article.

Submission queues are a contiguous host memory region acting as a circular buffer. Each submission queue entry is 64 bytes. The host notifies the controller of new commands by filling out the next entry in the queue and then writing to a per-queue doorbell in the controller's MMIO register space. In `nvme(4)`, this is done in `nvme_qpair_submit_tracker()`:

Completion queues are a separate contiguous host memory region with 16-byte entries. The controller notifies the host about completions by filling out the next entry in the completion queue. This entry contains both the submission queue's ID and the per-queue command ID that the host used when submitting the command. The controller then interrupts the host and the driver can start processing completion entries. NVMe defines a phase bit in each completion queue entry to enable the host to determine which completion entries are valid. The controller will write this phase bit to 1 the first time through the queue, and then alternating between 0 and 1 through subsequent passes through the queue. The driver checks each completion queue entry's phase bit against the expected value to determine which entries contain new completions. In `nvme(4)`, this is done in `nvme_qpair_process_completions()`:

```

696     /* Copy the command from the tracker to the submission queue. */
697     memcpy(&qpair->cmd[qpair->sq_tail], &req->cmd, sizeof(req->cmd));
698
699     if (++qpair->sq_tail == qpair->num_entries)
700         qpair->sq_tail = 0;
701
702     wmb();
703     nvme_mmio_write_4(qpair->ctrlr, doorbell[qpair->id].sq_tdb,
704                         qpair->sq_tail);

```

July/August 2018 **37**

```

424     while (1) {
425         cpl = &qpair->cpl[qpair->cq_head];
426
427         if (cpl->status.p != qpair->phase)
428             break;
429

```

This method of completion queue processing enables the host to only read memory to determine which commands have been completed. This ensures there are no MMIO reads in the performance code path. It also takes advantage of CPU features such as Intel®’s Data Direct I/O Technology³ to place completion entries directly in last-level cache to optimize completion entry processing.

NVMe commands are further divided into two types – admin and I/O – which are handled by two different types of qpairs – also named admin and I/O. The queue processing described earlier applies identically to both admin and I/O qpairs, but the method of initializing them is quite different. Walking through how an NVMe controller is initialized will help with this understanding.

NVMe Controller Initialization

At a high level, controller initialization is split into two stages:

Controller reset – this stage is performed by the host using NVMe MMIO register reads and writes. Its primary functions are describing the parameters of the admin qpair (host memory addresses and queue sizes) and then toggling an enable bit (**CC.EN**). After the enable bit is toggled, the controller will start its internal initialization and set its ready bit (**CSTS.RDY**) when it is completed. The host waits for the ready bit to become set before proceeding to the next stage.

Controller setup – this stage is performed by the host using admin commands which can now be submitted on the admin qpair that was setup in the previous stage.

- Submit an **IDENTIFY** command with command dword 10 (**CDW10**) set to 1. This signifies the controller to return the **IDENTIFY** data associated with the controller.
- Submit **CREATE_IO_CQ** and **CREATE_IO_SQ** commands to construct I/O qpairs. The number of I/O qpairs allocated typically equals the minimum of the number of CPU cores on the system and the maximum number of I/O qpairs supported by the controller. This will be described in more detail later in this article.
- Submit an **IDENTIFY** command for each namespace reported by the controller’s **IDENTIFY** data. The key piece of data in the namespace’s **IDENTIFY** data is the size and format of the namespace. The namespace is specified in the **IDENTIFY** command using the **NSID** field which always starts at 1 (there is never a namespace 0)!

I/O Queue Allocation

I/O qpair allocation is a key feature of the **nvme(4)** driver. With many I/O qpairs at our disposal, we can ideally allocate a separate qpair per CPU core. This enables threads on each CPU core to submit I/O commands without synchronization with threads running on other CPU cores. It also enables binding completion interrupts to the CPU core where the I/O was submitted to improve cache locality.

Practically, there are a couple of reasons why a per-core qpair may not be possible. First, while the NVMe specifications allow for up to 65,535 I/O qpairs per controller, most NVMe SSDs allow far fewer I/O qpairs – sometimes fewer than 32. Second, there may be a limited number of interrupt vectors available on the system. This latter limitation was exposed most frequently on systems with many NVMe SSDs and multi-queue NICs but should now be much less likely to occur thanks to some SMP improvements that are now in FreeBSD 11.4

For cases where a qpair cannot be allocated per CPU core, **nvme(4)** will allocate as many qpairs as it can, and then associate each qpair with multiple CPU cores. A mutex is used to synchronize access to the qpair – not only between different threads submitting I/O, but also with the qpair’s completion handler.

I/O Submission

An NVMe command primarily consists of the following and is represented by FreeBSD’s struct **nvme_command** found in **/usr/include/dev/nvme/nvme.h**:

- An 8-bit opcode – admin and I/O opcodes overlap but are easily differentiated based on its submission queue type
- A 16-bit command ID – this command ID must be unique among any other commands submitted on the same queue
- Namespace identifier – primarily used for I/O commands, but some admin commands such as `IDENTIFY` also use this field; note that this infers any I/O queue and submit I/O to any namespace
- Two Physical Region Page (PRP) entries

PRP is NVMe's version of scatter-gather lists (SGL). Instead of scatter-gather elements which typically specify a start address and length, PRP entries only specify a start address. The length is inferred by the distance of the start address from the next page, and the overall size of the I/O command. It also has limitations such as all PRP entries except the first must start on a 4KiB boundary, and all entries except the first and last must describe exactly a 4KiB length buffer. Fortunately, these limitations do not affect FreeBSD, since the FreeBSD I/O stack always uses virtually contiguous buffers which can always be represented by PRP.

An astute reader will notice that an NVMe command only has two PRP entries – so how do we represent the I/O buffers for large commands? In these cases, the second PRP entry points to a list of other PRP entries. But this means that for large I/O, we need buffers for the PRP lists that are mapped for DMA.

`nvme_request` and `nvme_tracker`

There are two considerations for PRP lists:

- PRP list buffers need to be mapped for DMA, so PRP list buffers are allocated and mapped when the qpair is allocated. This avoids mapping the PRP list in the performance path.
- PRP list buffers can become large in certain environments. The size of a PRP list buffer is proportional to MAXPHYS. FreeBSD's default MAXPHYS is 128KiB which results in a maximum PRP list of 256 bytes. Netflix however deploys with MAXPHYS of 1MiB which results in 2KiB PRP list buffer size. So, we need to be judicious in how many PRP lists are allocated per qpair. The `nvme` driver automatically limits the request size to the smaller of MAXPHYS and 2MiB, the largest buffer that fits in the two PRPs pages available for the SG list.

`nvme(4)` defines two different structures – `struct nvme_request` and `struct nvme_tracker`. When an I/O qpair is allocated, 128⁵ `nvme_trackers` are allocated, along with a PRP list buffer for each `nvme_tracker`. This represents the maximum number of I/O that can be outstanding on the qpair at any given time.

`struct nvme_request` represents a command that has been requested by the caller. The simplest case is `nvme_ns_cmd_read()`:

```

32 int
33 nvme_ns_cmd_read(struct nvme_namespace *ns, void *payload, uint64_t lba,
34     uint32_t lba_count, nvme_cb_fn_t cb_fn, void *cb_arg)
35 {
36     struct nvme_request *req;
37
38     req = nvme_allocate_request_vaddr(payload,
39         lba_count*nvme_ns_get_sector_size(ns), cb_fn, cb_arg);
40
41     if (req == NULL)
42         return (ENOMEM);
43
44     nvme_ns_read_cmd(&req->cmd, ns->id, lba, lba_count);
45
46     nvme_ctrlr_submit_io_request(ns->ctrlr, req);
47
48     return (0);
49 }
```

Here the caller is requesting to read data from the namespace into the buffer "payload". First, `nvme_allocate_request_vaddr()` allocates an `nvme_request` structure with the parameters specified by the caller. Next, the `nvme_command` structure is populated by `nvme_ns_read_cmd()`. Note that `nvme_command` is a 64-byte submission queue entry, but at this point we are only preparing the

submission queue entry – it will get copied later into the actual submission queue. Finally, we call `nvme_ctrlr_submit_io_request()`.

```
1207 void
1208 nvme_ctrlr_submit_io_request(struct nvme_controller *ctrlr,
1209     struct nvme_request *req)
1210 {
1211     struct nvme_qpair      *qpair;
1212
1213     qpair = &ctrlr->iog[curcpu / ctrlr->num_cpus_per_iog];
1214     nvme_qpair_submit_request(qpair, req);
1215 }
```

Here is where we pick the qpairs based on the current CPU. Now that we know which qpairs to use, we call `nvme_qpairs_submit_request()`. This function checks if there are any available `nvme_trackers`. If there are, it calls `nvme_qpairs_submit_tracker()` which we saw earlier. If not, it puts the `nvme_request` in an STAILQ. Later, once some I/O are completed, this STAILQ will be checked and `nvme_requests` are resubmitted.

NVMe Namespaces

The NVMe specification allows the logical partitioning of drives into individual namespaces. A namespace is nothing more than a collection of blocks, addressed 0..N-1. These namespaces may be fully provisioned or thinly provisioned (and in fact, hardware or hypervisors that emulate NVMe drives often hide these details from the host). Namespaces also provide ways to independently assign attributes to those namespaces. One namespace may be used to store the OS. The data rarely changes, usually has few writes, but a lot of reads. Another namespace may contain a transaction log which is read infrequently but written in a mostly append I/O pattern. Still another may contain data that's rewritten all the time and very hot. The firmware on the NVMe drive can use these attributes to optimize NAND storage. For cold data, like the OS, the firmware may place it into cells that have low wear and are storing 3 bits per cell to maximize data density. These cells often have excellent long-term retention. For very hot data, the drive may choose to use cells that are more worn and may store a lot of it in 1-bit-per-cell to maximize speed. While the worn cells cannot retain data as long, they are ideal for hot data because the data won't be stored for long anyway, so any deficiency in longevity will not hinder the drive's performance.

FreeBSD does not yet support namespace management – for example, creating and deleting namespaces and specifying attributes for those namespaces. There is active work in this area within the FreeBSD community however and should make it in time for FreeBSD 12.

Managing NVMe Drives

The primary utility for listing and configuring NVMe controllers and namespaces is `nvmecontrol(8)`. The most basic `nvmecontrol` subcommand is “`nvmecontrol devlist`” which provides a brief summary of the each NVMe controller and its `namespace(s)`.

```
%sudo nvmecontrol devlist
nvme0: ORCL-VBOX-NVME-VER12
    nvme0ns1 (1024MB)
```

Additional `nvmecontrol(8)` subcommands include:

- “`nvmecontrol identify`” for providing details on NVMe controllers and namespaces based on information from the NVMe `IDENTIFY` command
- “`nvmecontrol logpage`” for reading log pages from an NVMe controller; specification-defined log pages for Errors, Health/SMART, and Firmware Slots have handlers to translate the log page into a human-readable format
- “`nvmecontrol firmware`” for downloading and/or activating different firmware images on an NVMe controller
- “`nvmecontrol perftest`” for running a low-level performance test from the `nvme(4)` driver itself

- “nvmecontrol reset” to issue a controller-level reset to an NVMe controller
- “nvmecontrol power” to change the power state or specify a workload hint for an NVMe controller
- “nvmecontrol wdc” to perform options specific to WDC NVMe SSDs
- camcontrol(8) can currently be used to list namespaces when `nda(4)` is in use, but other functionality such as NVMe identify or firmware download are not yet plumbed.

One shortcoming of `nvmecontrol(8)` is mapping an NVMe controller or namespace to its associated `nvd` or `nda` entry. The best way to do this currently is associating serial numbers between “geom disk list” and “`nvmecontrol identify`”. As `camcontrol(8)` gains more NVMe functionality, this issue will be mitigated.

Summary

NVM Express provides a modern, high performance storage interface well suited to today’s CPU architectures – and FreeBSD is well-positioned to take advantage of it. Integrating NVMe support with CAM, as well as future support for namespace management and NVMe SSD hotplug will further improve on FreeBSD’s NVMe capabilities. Next time you watch Netflix, you will hopefully know a little bit more about how those bits ended up on your screen! ●

¹<https://www.intel.com/content/www/us/en/solid-state-drives/optane-ssd-900p-brief.html>

²<https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf>

³<https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html>

⁴https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=199321

⁵128 is the default. This number can be modified with the `hw.nvme.io_trackers` tunable.

Jim Harris is a Principal Software Engineer in Intel’s Data Center Group and was granted his FreeBSD source commit bit in 2011. He is the original author of the FreeBSD `nvme` and `nvd` drivers as well as the `nvmecontrol` management utility. Jim has also helped bring libraries and tools such as DPDK and Intel VTune Amplifier to FreeBSD and ported the FreeBSD `nvme` driver to userspace as part of his current role as Storage Performance Development Kit software architect.

Warner Losh is a Senior Software Engineer at Netflix, where he optimizes FreeBSD’s storage system for video delivery servers. He has been a FreeBSD contributor for over 20 years and is currently serving his sixth term on the FreeBSD core team. Warner has improved a number of systems—most recently the boot loader—in FreeBSD. Prior to Netflix, he produced flash drives and measured atomic clocks for accuracy. His code still measures some of the clocks that create UTC!

SUBSCRIBE TODAY



FreeBSD
JOURNAL®

Go to www.freebsdfoundation.org
1 yr. \$19.99/Single copies \$6.99 ea.





WeGetletters

by Michael W. Lucas

Hi Michael,

We were brainstorming column ideas for the FreeBSD Journal, and Kode Vicious suggested that you might be willing to handle a "Letters" column for us. People would submit their questions to the Journal, and you'd answer them for us. Any chance you'd be interested?

Best,

George V Neville-Neil
FreeBSD Foundation President

Hi George,

This is a terrible idea. It's just awful. This is the Internet age. Nobody reads letters columns, advice columns, or anything like that. We have Stack Exchange, and all kinds of places for people to beg for advice.

FreeBSD has a whole bunch of places where users can go to get specific help. Help ships with the system, in the man pages. Where a bunch of Unix-like operating systems made this absurd decision to bundle manual pages separately, FreeBSD ships with the manual. Actually, you can't not install the manual. You could build a FreeBSD that doesn't include the manual, of course, but to do that means reading a whole bunch of man pages.

People say that the manual isn't a tutorial, and they're right. That's why FreeBSD has the Handbook and a whole bunch of articles. Unlike the man pages, you can choose to not install those on a FreeBSD host. You can browse all of the documentation online at <https://docs.freebsd.org>, though.

New users can start with the FAQ (Frequently Asked Questions) file, which contains literally dozens of questions and answers. It goes into everything from hardware compatibility to ZFS, and while some of the gags in "The FreeBSD Funnies" have aged dreadfully—nothing scratches in memory banks these days, they fixed that bug back in 1996—the rest of the document is rock-solid. Looking at it now, I realize just how useful it is to new users. I still remember that feeling of enlightenment when I understood why du(1) and df(1) give different answers for disk space usage. Setting aside an hour to read the FAQ will give new users that enlightened feeling over and over again.

Then there's the Handbook. It's broken up by tasks. If a user's question has a little more depth than what's in the FAQ, the Handbook is there for you. Some of the material orients the reader, and is well worth reading so that new FreeBSD administrators understand why there's so much in /usr/local when everybody else just dumps everything in /etc and /bin.

Plus there's all sorts of FreeBSD-related sites these days. Even my blog has some FreeBSD tutorials on it.

If anyone did write in for help, it would be because they didn't use these resources.

—ml

Michael,

Not necessarily. People do have problems that aren't yet documented. We really think a letters column could be useful addition to the Journal, and that you're the right person to write it.

—Best, George

George,

Okay, let's talk about those folks who have issues that truly aren't in the Handbook.

Back when I started with FreeBSD, you got help via the FreeBSD-questions@FreeBSD.org mailing list. And it's still around today. The people on that list want to answer questions. They subscribe specifically so they can help people with their issues. Those brave people volunteer their time to answer user questions. What can I do that those heroes can't?

For those young punks who've forgotten how email works, there's a FreeBSD forum at <https://forums.freebsd.org>. Unlike the mailing list, the forums are broken up by category. Users can delve into detailed discussions of installation, storage, hardware, packages, or whatever. Whenever I look at the forums, I find interesting discussions.

There's a quarter century of problem-solving in the mailing list archives. What can I say that hasn't been said many times over?

These channels are really suitable for issues with particular hardware. The Handbook and FAQ are permanent fixtures in the FreeBSD ecosystem—they've been around for decades. But if some chipset in your brand-new knock-off laptop is caus-

ing you grief, you can search the mailing list or the forum to see if anyone else has that same issue with that hardware.

Users who can't be bothered to DuckDuckGo the mailing list archives or search the forums certainly aren't going to bother composing a coherent letter to me.

—ml

Michael,
Seriously, there's people out there who have problems that aren't in the Forums or mailing list archives yet. You really could help them. When they see how helpful you are, it might even encourage them to buy your books.

—Best, George

Dang it, George, you just don't give up, do you?

Okay, fine. Let's walk this through.

A user has a problem. A truly unique problem, that doesn't appear anywhere in the mailing list archives, the forums. The only reference on the Internet to a problem even vaguely like this is on a darknet site and in Siberian. They're sincerely and honestly in trouble.

Before anyone could help this user, they'd need to describe their problem in a useful way. This means they'd have to send a complete description of the problem. Most people who compose a request for help can't be bothered to give the output of "uname -a" and a copy of dmesg.boot. They can't trouble themselves by giving actual error output or the contents of /var/log/messages. Or they "helpfully" strip out stuff they think is irrelevant, like the messages saying "PHP is dumping core" that appear all through their web server logs.

And that's another thing. People want help with stuff that has no relevance to FreeBSD. They know it has nothing to do with FreeBSD. And yet they send a message to a FreeBSD mailing list? I mean, that's just rude.

And speaking of rudeness—would it hurt people to be polite when they ask for help? Anyone on the mailing list or the forum who takes time to help a user is volunteering their own time. They have better things to do than to put up with your tantrum. I mean, I get that computers can really torque people off. I myself have more than once stood on a rooftop and screamed foul obscenities at the buffer cache—who hasn't? But there's no need to take that out on someone who's trying to help you.

Most often, the mere act of writing the problem description is enough to make my own brain to solve the problem.

And nothing short of high voltage would encourage people to buy my books.

So, no. Let users with trouble go to the mailing lists or the Forums. I have enough to do.

—ml

Michael,
We'll only send good letters. I promise.
—Best, George

No. No, no, no.

NO.

Do you have any idea how many books I still have to write in my lifetime?

Ain't gonna. Can't make me.

—ml

We'll pay you in gelato.

—George

George,

Curse you. I'm in.

But tell Kode Vicious that if he drops my name again, he's going home in a bucket.

—ml

Questions?

Contact letters@freebsdjournal.org.

Letters will be answered in the order that they enlighten or amuse the columnist.

MICHAEL W. LUCAS has been a sysadmin for over twenty years. His latest books include *SSH Mastery, 2nd edition*, *Ed Mastery*, the third edition of *Absolute FreeBSD*, and *git commit murder*.

Learn more at <https://mwli.io>.

SVN UPDATE

by Steven Kreuzer

Life keeps getting in the way, so I have not been able to keep up with FreeBSD's fast-paced development. One day the sun and moon aligned and not only were both my daughters sound asleep, but I also found myself with enough energy to not only keep my eyes open, but also form cohesive thoughts. Not knowing what to do with all this free time I had on my hands, I did what any normal person would do and started to dig through the subversion logs. Imagine my surprise when I discovered that FreeBSD now supports pNFS, has a brand-new TCP congestion algorithm, and cron even gained new functionality!

Merge the pNFS server code from projects/pnfs-planb-server into head—

<https://svnweb.freebsd.org/changeset/base/335012>

A pNFS service separates the Read/Write operations from all the other NFSv4.1 Metadata operations. It is hoped that this separation allows a pNFS service to be configured that exceeds the limits of a single NFS server for storage capacity and/or I/O bandwidth. It is possible to configure mirroring within the data servers (DSs) so that the data storage file for an MDS file will be mirrored on two or more of the DSs. When this is used, failure of a DS will not stop the pNFS service and a failed DS can be recovered, once repaired, while the pNFS service continues to operate. Although two-way mirroring would be the norm, it is possible to set a mirroring level of up to four, or the number of DSs, whichever is less. The Metadata server will always be a single point of failure, just as a single NFS server is.

Permit sysctl(8) to set an array of numeric values for a single node—

<https://svnweb.freebsd.org/changeset/base/330711>

Most sysctl nodes only return a single value, but some nodes return an array of values (e.g., kern.cp_time). sysctl(8) understands how to display the values of a node that returns multiple values (it prints out each numeric value separated by spaces). However, until now, sysctl(8) has only been able to

set sysctl nodes to a single value. This change allows sysctl to accept a new value for a numeric sysctl node that contains multiple values separated by either spaces or commas. sysctl(8) parses this list into an array of values and passes the array as the "new" value to sysctl(2).

Bring in the TCP high-precision timer system (tcp_hpts)—

<https://svnweb.freebsd.org/changeset/base/332770>

It is the forerunner/foundational work of bringing in both Rack and BBR, which use hpts for pacing out packets. The feature is optional and requires the TCPHPTS option to be enabled before the feature will be active. TCP modules that use it must assure that the base components compile in the kernel in which they are loaded.

Bring in a new refactored TCP stack called Rack—

<https://svnweb.freebsd.org/changeset/base/334804>

RACK ("Recent ACKnowledgment") is a time-based, fast-loss detection algorithm for TCP. RACK uses the notion of time instead of packet or sequence counts to detect losses for modern TCP implementations that can support per-packet timestamps and the selective acknowledgment (SACK) option. It is intended to replace the conventional DUPACK threshold approach and its variants, as well as other nonstandard approaches.

AF_UNIX: make unix socket locking finer grained—

<https://svnweb.freebsd.org/changeset/base/333744>

This change moves to using a reference count across lock drop/reacquire to guarantee liveness. Currently sends on unix sockets contend heavily on read locking the list lock. unix1_processes in will-it-scale peaks at 6 processes and then declines. With this change, we see a substantial improvement in number of operations per second with 96 processes.

Fix spurious retransmit recovery on low-latency networks—

<https://svnweb.freebsd.org/changeset/base/333346>

TCP's smoothed RTT (SRTT) can be much larger than an actual observed RTT. This can be due to hz restricting the calculable RTT to 10ms in VMs or

SVN **UPDATE** continued

1ms using the default 1000hz or simply because SRTT recently incorporated a larger value. If an ACK arrives before the calculated badrxtwin (now + SRTT): $tp->t_badrxtwin = ticks + (tp->t_srtt >> (TCP_RTT_SHIFT + 1))$; we'll erroneously reset snd_una to snd_max. If multiple segments were dropped and this happens repeatedly, the transmit rate will be limited to 1MSS per RTO until we've retransmitted all drops.

Boost thread priority while changing CPU frequency— <https://svnweb.freebsd.org/changeset/base/33325>

Boost the priority of user-space threads when they set their affinity to a core to adjust its frequency. This avoids a situation where a CPU-bound kernel thread with the same affinity is running on a down-clocked core and will “block” powerd from up-clocking the core until the kernel thread yields. This can lead to poor performance and to things potentially getting stuck on Giant.

Import the netdump client code— <https://svnweb.freebsd.org/changeset/base/33283>

This is a component of a system which lets the kernel dump core to a remote host after a panic, rather than to a local storage device. The server component is available in the ports tree. netdump is particularly useful on diskless systems. The netdump(4) man page contains some details describing the protocol. To use netdump, the kernel must have been compiled with the NETDUMP option.

Improve VM page-queue scalability—

<https://svnweb.freebsd.org/changeset/base/332974>

Currently both the page lock and a page-queue lock must be held in order to enqueue, dequeue or requeue a page in a given page queue. The queue locks are a scalability bottleneck in many workloads. This change reduces page-queue lock contention by batching queue operations. To detangle the page and page-queue locks, per-CPU batch queues are used to reference pages with pending queue operations. The requested operation is encoded in the page's aflags field with the page lock held, after which the page is enqueued for a deferred batch operation. Page-queue scans

are similarly optimized to minimize the amount of work performed with a page-queue lock held.

Add new functionality and syntax to cron(1) to allow jobs to run at a given interval—

<https://svnweb.freebsd.org/changeset/base/334817>

The practical goal here is to avoid overlap of previous job invocation to a new one, or to avoid too short interval(s) for jobs that last long and don't have any point of immediate launch soon after the previous run. Another useful effect of interval jobs can be noticed when a cluster of machines periodically communicates with a single node. Running the task time-based creates too much load on the node. Running interval-based spreads invocations across machines in cluster.

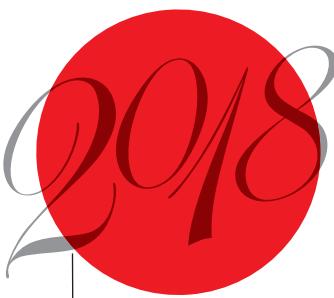
Load balance sockets with new SO_REUSEPORT_LB option— <https://svnweb.freebsd.org/changeset/base/332894>

This patch adds a new socket option, SO_REUSEPORT_LB, which allows multiple programs or threads to bind to the same port and incoming connections will be load balanced using a hash function.

Add the TCP Blackbox Recorder— <https://svnweb.freebsd.org/changeset/base/331347>

The TCP Blackbox Recorder allows you to capture events on a TCP connection in a ring buffer. It stores metadata with the event. It optionally stores the TCP header associated with an event (if the event is associated with a packet) and also optionally stores information on the sockets. It supports setting a log ID on a TCP connection and using this to correlate multiple connections that share a common log ID. You can log connections in different modes. If you are doing a coordinated test with a particular connection, you may tell the system to put it in mode 4 (continuous dump). Or, if you just want to monitor for errors, you can put it in mode 1 (ring buffer) and dump all the ring buffers associated with the connection ID when we receive an error signal for that connection ID. You can set a default mode that will be applied to a particular ratio of incoming connections. You can also manually set a mode using a socket option.

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retro-computing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.



AUGUST/SEPTEMBER

BY ANNE DICKISON & DRU LAVIGNE

Events Calendar

The following BSD-related conferences will take place in August and September of 2018.

27th USENIX Security Symposium

AUGUST 15-17, 2018
BALTIMORE, MD

USENIX Security '18 • August 15-17 • Baltimore, MD

<https://www.usenix.org/conference/usenixsecurity18> • The 27th USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security and privacy of computer systems and networks.



FreeBSD

BSDCam 2018 • August 15-17 • Cambridge, UK

<https://bsdcam.cl.cam.ac.uk> • The 2018 Cambridge FreeBSD DevSummit (BSDCam) will take place August 15-17, 2018, in Cambridge, UK. The event is run "un-conference style" in that we brainstorm the actual session schedule on the first morning, with a focus on interactive topics that reflect the interests and exploit the knowledge of the attendees.

Write the Docs+Open Help • August 18-22 • Cincinnati, OH

<http://www.writethedocs.org/conf/cincinnati/2018/> A conference focused on all things related to software documentation.

FOSSCON

FOSSCON • August 25 • Philadelphia, PA

<https://fosscon.us> A Free and Open Source software conference held annually in Philadelphia PA, FOSSCON brings together users of Free and Open Source Software from a wide variety of fields.

OpenZFS Developer Summit • Sept 10-11 • San Francisco, CA



http://open-zfs.org/wiki/OpenZFS_Developer_Summit • The sixth annual OpenZFS Developer Summit will take place September 10-11, 2018 in San Francisco. As with previous years, the goal of the event is to foster cross-community discussions of OpenZFS work and to make progress on some of the projects that have been proposed. This 2-day event consists of a day of presentations and a 1-day hackathon.

EuroBSDCon 2018 • Sept 20-23 • Bucharest, Romania



<https://2018.eurobsdcon.org> • EuroBSDcon is the European annual technical conference gathering users and developers working on and with 4.4BSD (Berkeley Software Distribution) based operating systems family and related projects.

Grace Hopper 2018 • Sept 26-28 • Houston, TX



<https://ghc.anitab.org/2018-attend/> • The Grace Hopper Celebration is the world's largest gathering of women technologists. It is produced by AnitaB.org and presented in partnership with ACM.